Universidad Carlos III de Madrid
Escuela Politécnica Superior

Ingeniería de Telecomunicación
Proyecto Final de Carrera

"Desarrollo de widgets para la plataforma Maemo"

Autor: Sergio Manuel Galán Nieto
Tutor: Abelardo Pardo Sánchez

Agradecimientos

A mi padre y a mi madre por educarme, apoyarme y por quererme tanto: es difícil crecer en mejor entorno. A mi hermana porque es un consuelo saber que ahí fuera siempre habrá alguien como yo. A mi abuelo porque aunque no entienda muy bien a que me dedico se siente muy orgulloso de que su nieto sea ingeniero.

A Abelardo por guiarme, por ayudarme a concretar, por responder siempre pronto con el lío que tiene, por haberse leído a conciencia lo que le he enviado, y por haber aguantado la pasimonia con la que me he tomado todo esto. También a la cátedra Nokia por la financiación y los juguetes.

A Marina por ponerme un deadline que ha sido crucial y porque ha convertido mi vida en una sucesión de días preciosos que hacen que despertarse realmente merzca la pena.
No puedo no mencionar a Paco, Nacho y Victor que son como hermanos: El tiempo pasa pero siempre están ahí para todo aunque cada uno estemos en un lado. Y gracias también al resto de mis amigos que no son muchos, pero que no podría ponerlos todos (Jose, Gema, Jaime, Pello, Pablo, Sara y toda la tribu del A01 y alrededores, todos los amigos jaeneros y jaenitas ...) Aunque a veces parezca un soso, realmente valoro cada rato y cada conversación que hemos tenido juntos.
También un recuerdo para todos los que han sufrido y terminado una práctica a mi lado: A pesar de todo ahora parecen buenos momentos... que no deben volver.

Resumen

En este proyecto se presentan una serie de programas que funcionan incrustados en el escritorio del sistema operativo Maemo 2008 -disponible para dispositivos móviles Nokia N800 y N810. Estos programas son conocidos como widgets y realizan funciones muy concretas. Los cinco widgets desarrollados se encargan de obtener información desde un servidor .LRN, proporcionar una interfaz para los servicios web Flickr y Twitter y ofrecer servicios basados en la posición: Alertar de tareas que deban realizarse al pasar cerca de un lugar y recomendar servicios de ocio cercanos. Como parte imprescindible, además, se han debido de estudiar y depurar las diferenes formas de desarrollo disponibles para esta plataforma móvil.

Abstract

This project shows several small programs which work attached in the desktop of the Maemo 2008 OS- available for Nokia N800 and N810 Internet Tablets. These programs are known as "widgets" and their functions are well defined and reduced. Each one of the five widgets works getting information from .LRN platform, offering a desktop interface for Flickr and Twitter web services, or providing solutions based on position: The first one alerts from tasks that must be done close to some place and the other one recommends services close to the user current city area. It was also essential in this project to study and to polish the several developing process available for this mobile platform.

Índice

Índice de figuras

1. Introducción

Este proyecto presenta una serie de programas simples orientados a facilitar la interacción entre dispositivos móviles y servicios de Internet, los denominados widgets. Estos widgets se han desarrollado para los terminales N800 y N810 de Nokia, una plataforma móvil basada en el sistema operativo (S.O.) Linux.

1.1. Motivación del proyecto

Desde su popularización masiva a finales de los años 90 hasta aproximadamente el año 2007, el principal método de acceso doméstico a los servicios de Internet fue el de utilizar un PC (Ordenador Personal) con una conexión fija. Sin embargo, actualmente el acceso a la web desde el PC va cediendo parte de su importancia al acceso desde los dispositivos móviles. En países como Japón esta tendencia se encuentra mucho más extendida y el teléfono móvil es la principal vía de acceso a la web, algo que se prevé que también suceda en un futuro en los países europeos.

El camino para pasar de una web atada al PC a una web ubicua ha venido marcado por dos rutas tecnológicas paralelas:

- En el lado de las tecnologías web, se ha producido un traslado gradual de las páginas web con contenido y presentación unidas en un fichero de lenguaje de Hypertexto (HTML) a una separación entre ambos. Por un lado la información, cada vez más metaetiquetada semánticamente, y por otro lado diversas formas de mostrar esa información, principalmente usando Hojas de Estilo en Cascada (CSS).
- En el lado de los terminales, éstos cada vez cuentan con procesadores más potentes, grandes pantallas (frecuentemente táctiles) y mayores velocidades de transferencia.

Además, la Web ha mutado de un simple espacio para publicar información a una nube de cómputo en la que se ejecutan aplicaciones interoperables que se muestran al usuario, normalmente, a través del navegador.

Es en este contexto donde cobran sentido los widgets para dispositivos móviles. Los widgets son pequeños programas que se ejecutan en el escritorio del usuario y que sirven para realizar pequeñas tareas: diccionarios, búsqueda, muestra de información personalizada, etc. Normalmente utilizan la red para funcionar, con lo que pueden considerarse una interfaz alternativa al navegador. En los terminales móviles este tipo de programas pueden ser especialmente útiles puesto que con un tamaño de pantalla menor que el de un PC, hay que encontrar formas específicas de presentar la información optimizando al máximo el espacio.

Los widgets suelen ejecutarse sobre una "plataforma de widgets". Estas plataformas normalmente ofrecen formas sencillas de desarrollo usando tecnologías web como CSS JavaScript y Lenguajes de etiquetas (XML). De este modo, sin tener grandes conocimientos de programación es posible escribir widgets que son interpretados por la plataforma correspondiente.

En este proyecto se han desarrollado widgets para los dispositivos que soportan la versión 2008 de la plataforma Maemo de Nokia, los terminales N800 y N810. La elección del dispositivo vino impuesta por el contexto en el que este proyecto fue llevado a cabo: una beca de desarrollo de la Cátedra Nokia del Departamento de Ingeniería Telemática. Estos dispositivos no son teléfonos móviles sino una variación de asistente digital personal (PDA) orientado a la navegación por Internet, y que funciona usando el sistema operativo Linux.

En la versión 2008 del sistema operativo Maemo, los widgets tomaban importancia en el escritorio bajo el nombre de "Hildon Desktop Home Plugin" (HDHP), y por defecto bajo este formato aparecen aplicaciones como una radio o un lector de fuentes web. Estas aplicaciones se programan de forma tradicional, usando un lenguaje de programación como C o Python, ya que Maemo 2008 no cuenta con una plataforma de widgets propia.

Una vez estudiada la viabilidad técnica de desarrollar widgets para estos dispositivos, se comenzó el trabajo explorando las posibilidades de interacción con la plataforma educativa en la que el departamento de Ingeniería Telemática trabaja: .LRN. Tras unas primeras pruebas, aprendiendo a programar la plataforma y a programar .LRN, se concluyó que .LRN no estaba bien preparado para su interrelación con software externo y se pasó a trabajar con otros servicios web que directamente ofrecían una API para esto. Así en esta segunda etapa definida por la interrelación con servicios de la Web 2.0 fueron creados widgets para acceso a Flickr y a Twitter, servicios de almacenamiento de fotografías y de microblogging respectivamente.

Tras pasar a trabajar con terminales de nueva generación N810, se pensó en darle utilidad al dispositivo GPS que incorporan. De esta forma se plantearon los widgets que componen la tercera etapa de desarrollo del proyecto, la etapa de *locative media*. Así, se idearon aplicaciones que pudieran mejorar el día a día de un urbanita convencional y surgieron un recomendador de servicios y un widget para avisar de que hay que hacer determinada tarea al pasar por un sitio en concreto.

1.2. Objetivos

Cada widget tiene unos objetivos particulares que se encuentran descritos en su apartado correspondiente. En general los objetivos del proyecto son:

- Estudiar la viabilidad del desarrollo de widgets de escritorio sobre la plataforma Maemo.
  Existen widgets incluidos en la versión 2008 del sistema operativo de Maemo, pero antes de pasar al desarrollo de aplicaciones complejas hay que comprobar qué tipo de cosas pueden hacerse con la API que se ofrece y qué cosas no pueden hacerse.

- Encontrar formas de desarrollo óptimas para la plataforma Maemo.
  Uno de los problemas de trabajar con plataformas poco maduras como

ésta, es el escaso perfeccionamiento de sus entornos de desarrollo y la dificultad de encontrar un método sistemático que proporcione, en todo caso, resultados óptimos en cuanto a velocidad de desarrollo, depuración y calidad en el código. Por tanto además del desarrollo mismo es importante averiguar cómo llevarlo a cabo de la mejor forma posible.

- Realizar aplicaciones en formato widget que interactúen con servicios web.
  Las aplicaciones web actuales se caracterizan por permitir el acceso a ellas desde otras aplicaciones. En este caso se van a construir widgets que interactúen con servicios web conocidos y permitan acceder a un subconjunto definido de las funciones que se presentan vía web.

- Estas aplicaciones deben de quedar en un estado lo suficientemente terminado para que permitan su utilización por parte de un usuario.
  No se pretende una depuración completa que contemple todos los posibles errores ante cualquier tipo de acción del usuario (lo que sería una fase de producción) pero si un funcionamiento apropiado en la mayoría de los casos.

- Estudiar la interrelación entre .LRN y los widgets.
  Se desarrollará algún widget que obtenga y muestre información procedente del gestor de contenidos educativos .LRN

- Diseñar aplicaciones novedosas en la plataforma que hagan uso de la información geográfica.
  Un factor nuevo de las aplicaciones para móviles es que pueden ser ejecutadas en cualquier lugar. Disponer de información sobre este lugar posibilita el desarrollo de un nuevo tipo de aplicaciones ligadas al espacio.

1.3. Contenido de la memoria

La presente memoria se estructura en varios bloques. En primer lugar un estado del arte que describe cómo se encuentra actualmente, a nivel técnico y

comercial, el mundo del desarrollo de aplicaciones para dispositivos móviles y la importancia de los widgets dentro de cada plataforma móvil (sección 2). También se comenta la relación entre las aplicaciones de escritorio y las aplicaciones web y por último se incluye una descripción y algunos ejemplos de lo que se conoce como *locative media*. En definitiva, se describe el contexto técnico y comercial en el que se insertan las aplicaciones que componen este proyecto.

A continuación se incluye una descripción de la plataforma móvil sobre la que se trabaja, tanto a nivel software y hardware como a nivel comercial y de fuerza de la comunidad, algo muy importante en el mundo *open source* (sección 3).

La tercera parte comprende el proceso de adquisición del denominado *know-how* (o saber hacer). Es decir, empezar sin saber nada ni de la plataforma, ni del lenguaje de programación a usar, ni del entorno de desarrollo y terminar con conocimientos de todo lo mencionado. Esta parte tiene especial importancia en un proyecto en el que se va abriendo camino, puesto que el desarrollo de widgets para Maemo es un campo poco explorado fuera de la propia empresa Nokia (sección 4).

El bloque principal es el que describe en qué consiste cada uno de los programas que componen este proyecto (sección 5). El proceso de trabajo ha sido acumulativo. Escribiendo cada programa se han aprendido cosas que se incorporan en los siguientes, por eso en cada sección sólo se hace énfasis en los elementos nuevos que se han incorporado respecto a los anteriores programas. También para cada widget se describen los objetivos, los aspectos especialmente interesantes del desarrollo que se trata y una estructura general del código final.

Por último se encuentra la sección 6 con las pruebas realizadas a las versiones finales de los programas y una conclusión (sección 7) con los objetivos cumplidos y con las tareas que restan por hacer para obtener un software más completo y sólido de cara al usuario final.

Las instrucciones de instalación y de uso aparecen en los anexos junto con un glosario de términos.

2. Estado del arte

El presente proyecto utiliza tecnologías creadas para la web pero a las que se pretende sacar del navegador, concretamente para incrustarlas en el escritorio de un terminal móvil. Por eso en este estado del arte se van a tratar aspectos, tanto técnicos como de mercado, relacionados con las tecnologías web, con los dispositivos móviles y con los widgets en cada dispositivo: componentes básicos principales, características de uso, tipos de plataformas, etc.

2.1. La web actual

Desde la popularización de Internet en España a finales de los años 90 hasta hoy, el panorama que el usuario se encuentra al abrir su navegador ha cambiado radicalmente. En un principio el contenido que podía encontrarse estaba construido principalmente usando lenguaje HTML y era estático. Es decir, cualquier modificación requería editar un fichero HTML en el propio ordenador y después enviarlo mediante un protocolo de transferencia de ficheros (FTP) al servidor.

El camino hacia la creación y posterior colapso de la burbuja tecnológica vino acompañado de la popularización de los denominados "portales", los cuales suponen una visión de Internet totalmente contaría a la de la Web 2.0 que se explicará a continuación. Estos portales pretendían constituirse en puerta de entrada de un usuario a la red. Dentro del portal se proveería al usuario de servicios de noticias, tiendas, enlaces recomendados, buscador o correo. La participación del usuario quedaba seriamente restringida y era la empresa propiedad del portal la que generaba contenido o, en el mejor de los casos, apuntaba hacia contenido externo que seguía siendo mayormente estático.

Así pues la barrera técnica de subir contenido a la red provocaba tres limitaciones:

- Temporales: la información no se enviaba a la red en tiempo real.

- De contenidos: gran proporción de contenidos técnicos relacionados con las tecnologías de la información (IT) o generados por usuarios con conocimientos técnicos
- De uso: múltiples transiciones y esperas. Diseños poco claros que no facilitaban al usuario la navegación.

Sin embargo, existían algunas web que pueden considerarse precursoras de la actualidad. Por ejemplo slashdot.org fue creado en 1997 y a través de código ejecutable en los servidores permite a los usuarios enviar noticias y comentarios, es decir, algo casi idéntico a los blogs actuales.

Estas webs se mantienen desde entonces, al contrario que los portales que tras el colapso de la burbuja tecnológica desaparecieron o perdieron importancia. A pesar de este colapso en 2001, la web ha seguido creciendo y en la actualidad lo impregna todo y todos participa en la red: tecnologías web para el desarrollo de aplicaciones antes reservadas al sistema operativo, participación de los usuarios, colaboración, integración de la red en el escritorio, etc. Estas transformaciones tanto a nivel de uso como a nivel técnico son las que se exponen en los siguientes apartados.

2.1.1. Web 2.0

A veces un conjunto difuso de ideas y proyectos con puntos en común, sólo necesitan que alguien les de nombre para estallar y tomar fuerza. Fue Tim O'Reilly en 2004 quien presentó en público el término Web 2.0 [4]. En principio parecía un término de marketing que denominaba algo aún difuso: páginas web con una nueva estética mas limpia y orgánica, transiciones dinámicas similares a las aplicaciones de escritorio, participación de los usuarios, etc. Es decir un conjunto de tecnologías, diseño y percepción de los servicios que empezaban a suponer un salto visible respecto a todo lo anterior. En cuanto a tecnologías las piezas clave que se asocian con la Web 2.0 son:

XML (*Extensible Markup Language*)

Un lenguaje de etiquetas con el que podemos dotar de significado semántico a la información, en oposición al HTML que es tan sólo un lenguaje de presentación. Con XML se construyen web-services apoyados en SOAP, XML-RPC o se distribuye el contenido de los blogs usando RSS.

AJAX (*Asynchronous JavaScript And XML*)

Antes de AJAX, a cada acción del usuario, y sólo a cada acción, le correspondían una serie de peticiones y respuestas HTTP que provocaban la actualización de la página web. AJAX es un conjunto de tecnologías (XML y JavaScript) que permiten un desarrollo web interactivo: el usuario realiza su trabajo y de fondo, de forma asíncrona, el navegador comunica con el servidor para actualizar partes de la página. De esta forma se dota a las aplicaciones web de una interactividad similar a las aplicaciones de escritorio: ya no (siempre) hay que hacer click y esperar.

CSS (*Cascading Style Sheets*)

HTML como lenguaje para definir una capa de presentación se queda muy corto. En la tecnología web actual CSS es el lenguaje que permite dotar de riqueza gráfica al contenido que se muestra en los navegadores.

HTML (*HyperText Markup Language*)

Sigue presente aunque con un papel menos dominante que antes debido a la irrupción de las tecnologías anteriores. Su función como marcado presentacional prácticamente ha desaparecido y casi tan sólo se utiliza como marcado estructural, definiendo el propósito del texto: encabezamientos, palabras importantes, enlaces, etc.

HTTP (*HyperText Transfer Protocol*)

Es la base sobre la que se asientan todas las tecnologías web. Ha sido extendido en sucesivas versiones pero su principal aplicación se mantiene idéntica: a través de cabeceras en texto plano, permite la transferencia

de información entre un servidor y una serie de clientes siguiendo un protocolo de peticiones y respuestas.

Pero no es únicamente la tecnología la que define la nueva web. Su gran elemento diferenciador es el lugar que toma el usuario. Pasa de ser un consumidor de información a un generador. En la "Web 1.0" el paradigma eran los grandes portales que el usuario colocaba como página de inicio y desde donde se le guiaría por la gran confusión de contenidos de la red. En la Web 2.0 el paradigma lo empezaron a construir los blogs individuales, colaborativos, y los sitios como Flickr o Del.icio.us. Es decir, en la Web 2.0 la línea entre el productor y el consumidor de contenidos desaparece y el contenido de Internet se elabora en común.

2.1.2. Widgets: del escritorio al navegador y la vuelta al escritorio

Tecnológicamente en el centro de esta revolución se encuentra el navegador de Internet. Compañero diario del usuario en su paseo por la red, el navegador ha evolucionado hasta ser el software mas importante de los que se encuentran instalados en el sistema y ha engordado enormemente en cuanto a la cantidad de tecnologías que integra (estándares HTML cada vez más complejos, CSS, XML, Flash, JavaScript, etc.) Con estas tecnologías del lado del cliente y con un gran número de lenguajes de programación del lado del servidor, el navegador parece emerger como la interfaz universal tanto para la búsqueda de información como para la ejecución de aplicaciones.

Por otro lado las nuevas tecnologías web implican una división entre contenido y presentación. Ésto, junto con el uso masivo de diversos formatos XML para establecer APIs, ha provocado que mucha información accesible desde el navegador también pueda ser accedida desde terceras aplicaciones. En este mundo de formatos estandarizados para la transmisión de información es donde entra en juego la comodidad del usuario ¿hay servicios que puedan ser utilizados mejor desde una aplicación externa que desde el navegador?

Apple, en la versión 10.4 de su sistema operativo MAC OS X, introdujo el "Dashboard", una capa oculta entre el escritorio y las ventanas que se hace visible cuando el usuario la activa con un gesto especial. En esta capa viven los widgets, pequeñas aplicaciones que se encargan de facilitar tareas habituales, desde tomar notas, a consultar información de la Wikipedia o realizar una búsqueda en Google. La idea fue rápidamente adoptada por otros muchos sistemas operativos e incluso por dispositivos móviles: Microsoft los introdujo en su versión de Windows Vista, Yahoo ofrece widgets para MS Windows y Mac, y Nokia en su Serie60 cuenta con la plataforma Widsets. Normalmente estas aplicaciones están escritas en un lenguaje simple tipo XML con JavaScript y son interpretadas por la plataforma de widgets correspondiente. Su sencillez de programación las ha hecho abundantes y gratuitas.

La tendencia general es que cada vez más aplicaciones se encuentran en la red y menos en nuestro PC, lo que se denomina "Cloud computing". Sin embargo a la vez para ciertas tareas el hecho de tener que acudir a una página web es incómodo. Es aquí donde cobran más sentido los widgets.

2.2. Aplicaciones móviles

Mientras que el número de PCs en el mundo es de mil millones el número de teléfonos móviles es de cuatro mil millones [5][6]. En los países emergentes el número de personas con teléfono móvil es mucho mayor que el de personas con PC y en España la media de teléfonos por habitante es superior a uno. Pero, por supuesto un teléfono móvil hace tiempo que no es sólo un teléfono y de hecho se espera que se convierta en la principal plataforma de acceso a Internet. Los servicios de voz y mensajería son una *commodity* en cualquier terminal, y son las aplicaciones avanzadas las que aportan ventajas competitivas a unos terminales sobre el resto. Por eso actualmente nos encontramos en un periodo de desarrollo de novedosas aplicaciones para dispositivos móviles y de lucha entre las empresas del sector por colocar en posición dominante su plataforma, para lo que tratan de atraer al máximo número de desarrolladores posible.

2.2.1. La guerra de plataformas: Apple, Nokia, Google

En 2008 se vendieron 38 millones de *smartphones*. El gran presente y el enorme futuro de los dispositivos móviles hacen que ningún gran jugador de la industria tecnológica quiera quedarse fuera. Centrándose sólo en estos teléfonos móviles de gama alta, la cuota de mercado, por sistema operativo, a finales de 2008 se puede ver en la figura 1 [7]. Se aprecia un dominio total del sistema Symbian con casi un 50 % de la cuota total seguido del sistema de BlackBerry y del Windows Mobile de Microsoft. Estos son 3 participantes clásicos en la competición, pero en los últimos años la irrupción de Apple con iPhone y actualmente la de Google con su Android mantienen el mercado en un estado de gran dinamismo.

Pese a que Nokia controla las ventas, la dificultad de programación de sus terminales provoca que el número de aplicaciones no desarrolladas por Nokia para ellos sea limitado y sus dispositivos han perdido prestigio comparados con iPhone y Android. Para cambiar la situación, Nokia se ha hecho con la compañía desarrolladora de las librerías Qt: TrollTech. Estas librerías son multiplataforma y sencillas de utilizar si se comparan con Symbian+Serie60. La idea de Nokia es conseguir desarrollar aplicaciones que funcionen en cualquier tipo de dispositivo. La apuesta de Nokia por los widgets tiene el nombre de Widsets, una plataforma basada en Java Mobile Edition (JavaME), con lo que soporta teléfonos que no sean Symbian (pero no Maemo) y que permite construir pequeñas aplicaciones con código XML y CSS y un lenguaje script propio, el "Helium Script Lenguage". Desde Junio de 2009 la plataforma Widsets desaparece para integrarse en la tienda de aplicaciones online "OVI" y cede su lugar a WebRunTime (WRT). WRT es un motor de widgets que permite construir interfaces a partir de tecnologías web (CSS, XML y JavaScript) pero que además puede ser integrado en aplicaciones tradicionales Symbian/C++ con lo cual pueden construirse interfaces con tecnología web y a la vez acceder a la API propia del teléfono.

Apple es ahora mismo el referente de la industria en cuanto al desarrollo de aplicaciones para terminales móviles. Su tienda de aplicaciones ha generado

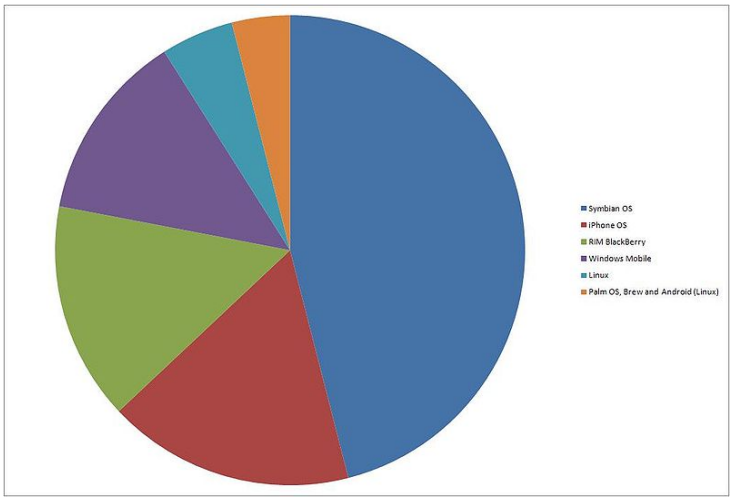

Figura 1: Cuota de mercado por S.O. en *smartphones* (Nov 2008) [7]

ya mil millones de descargas (abril 2009 [24]) y el resto de competidores se han apresurado a abrir sus propias tiendas. La primera visión que tuvo Apple de las aplicaciones para iPhone era cercana al concepto de widget. Apple no permitía desarrollar aplicaciones nativas y pretendía que existieran tan sólo aplicaciones basadas en web adaptadas a la pantalla de iPhone. De esta forma el navegador (Safari Mobile) se comporta como una plataforma de widgets. Posteriormente se permitió la existencia de aplicaciones nativas, más potentes y capaces de aprovechar la potencia del terminal.

No existe una plataforma de widgets de escritorio propia para iPhone, sin embargo los servicios de widgets online como Netvibes ofrecen una versión personalizada para iPhone [17]. El servicio de Plusmo [25], por contra, ofrece un software instalable en iPhone que incorpora un micronavegador a través del que se muestran sus widgets.

Google parte desde cero en este mercado pero su propuesta al frente de la plataforma *OpenHandset Alliance* ha recibido gran atención y se plantea como el principal rival de iPhone. Ofrece Android, una plataforma abierta que varios fabricantes de móviles pueden usar a la vez sin *royalties*. Ofrece una integración total con todas sus aplicaciones web y ofrece una API de desarrollo sencilla y potente. Muestra de ello es que hasta febrero de 2009

sólo existía un dispositivo móvil en el mercado con el sistema operativo de Google, pero el número de aplicaciones ya era de varios cientos. Los widgets en Android fueron incorporados a partir de la versión 1.5 del SDK, previamente no estaban documentados y sólo existían unos widgets simples creados por el equipo de desarrollo. Permiten mostrar información proveniente de otras aplicaciones de Android y su interfaz se construye mediante un archivo XML.

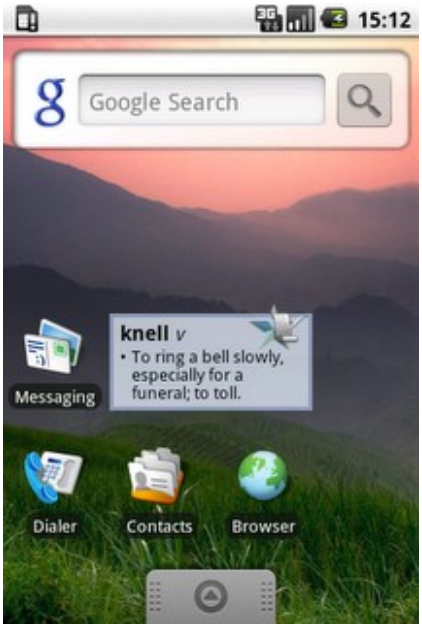

Figura 2: Widgets en Android

En esta guerra comercial Linux sólo se encuentra presente en algunos dispositivos en las capas inferiores, como en Android, y la plataforma Maemo que se trata en este proyecto, es más parecida a una PDA que a un teléfono. En el mundo *open source*, tras Maemo y Qt, el principal participante en el segmento de los *smartphones* es el de la comunidad OpenMoko que cuenta con un terminal (Free Runner) pero que aún se encuentra en estado beta.

2.2.2. Las nuevas aplicaciones móviles

Si bien muchos modelos anteriores incorporaban casi la misma potencia en cuanto a hardware, el iPhone de Apple marcó un punto de inflexión al ser el primer terminal móvil pensado para funcionar continuamente conectado a Internet. Esto ha provocado una avalancha de nuevas aplicaciones que además sacan partido de la ventaja principal del teléfono móvil, su ubiquidad. Podemos resaltar los siguientes aspectos:

- Nuevas formas de interacción: además del clásico teclado, las pantallas táctiles se están haciendo cada vez más presentes. Y debido a la incorporación de acelerómetros la entrada de órdenes gestuales también es relativamente común.
- Interfaces más amigables: el diseño de las nuevas aplicaciones saca partido de las nuevas formas de interacción y de pantallas mas amplias proporcionando transiciones más naturales y más facilidad de uso.
- Posicionamiento: bien mediante GPS, bien mediante triangulación de antenas, el terminal móvil puede saber donde se encuentra y las aplicaciones pueden utilizar esa información para enriquecer las aplicaciones con información relativa a la posición actual
- Interacción con el entorno: con todo lo anterior y además con una cámara incluida en casi cualquier teléfono actual, también son comunes tareas como leer códigos de barras, analizar fotografías o incluso combinar la imagen real con información extraída de la red (realidad aumentada).

Mezclando estos cuatro factores en distintas proporciones se obtiene una nueva generación de aplicaciones que están variado totalmente la forma de entender el software para telefonía móvil: de clones de las aplicaciones del PC a aplicaciones con su propia personalidad.

2.2.3. *Locative media*: mundo físico, mundo virtual

Dentro de las aplicaciones que se están desarrollando para teléfonos móviles quizás las más revolucionarias son las que utilizan la posición del usuario. Revolucionarias en el sentido de que no son traslación directa de aplicaciones disponibles en un PC, sino que hacen de la movilidad su gran baza.

Por primera vez, se crea un mapeado directo entre contenidos digitales y espacios físicos y de esta interrelación pueden surgir nuevas formas de entender el espacio urbano y también de utilizar los datos digitales. Aún el GPS no es común en los terminales móviles y ya existen varias aplicaciones interesantes, por ejemplo:

- ShopSavvy Permite escanear el código de barras de un producto e informa de en qué tiendas cercanas ese producto está mas barato [18]
- El diario The Guardian ofrece una plataforma abierta para utilizar su contenido en aplicaciones [19]. A través de esta plataforma el colectivo "Stamen" intenta que los lectores geoetiqueten el contenido del periódico para que los vecinos puedan buscar noticias relacionadas con su área.
- Parallel Kingdom es un juego de rol online para Android y iPhone que hace uso del GPS para colocar los elementos clásicos de un juego de rol sobre el mundo real [20]

3. La plataforma Maemo

Nokia cuenta en la actualidad con diversas familias de dispositivos móviles. Su principal apuesta es la de dispositivos de la plataforma Symbian/Serie60, un sistema operativo propietario diseñado principalmente por Nokia con los móviles/*smartphones* como objetivo. Por otro lado cuenta con la Serie40 para teléfonos móviles de bajas prestaciones. Y en una rama totalmente separada aparecen las Internet Tablet que no cuentan con funcionalidad de teléfono móvil y que son básicamente PDAs pensadas para navegar por Internet.

3.1. Familia de dispositivos. Hardware

El primer dispositivo de este tipo fue el Nokia N770 y data de 2005. En él se encuentra presente el hardware común a estos dispositivos:

- Procesador familia ARM en este caso de 252 Mhz
- Pantalla táctil de 800x480 y 16 bits
- Conexiones Wifi 802.11g, Bluetooth y USB
- Expansión de almacenamiento mediante tarjetas de memoria

Sus posteriores expansiones se han caracterizado por:

- N800:
  - Aumenta la memoria RAM de 128 a 256
  - Incluye cámara móvil
  - Aumento de la velocidad del procesador
- N810

Figura 3: Familia N de Nokia

- Presencia de teclado físico
- La memoria de almacenamiento interna se extiende hasta los 2 GB

Se aprecia que estos dos últimos dispositivos son muy similares y es por esto que las versiones actuales del sistema operativo Maemo funcionan en ambos, y no en la versión N770.
Pese a no disponer de conexión GSM o UMTS la conectividad a estas redes se supone en conjunción con un teléfono móvil al que se enlaza mediante Bluetooth. También duarante algunos meses existió una versión del N810 con conectividad Wimax, que fue eliminada en Junio de 2009 [26].

Durante su uso, este hardware se aprecia limitado para el software que incorpora puesto que el dispositivo adolece de ciertos problemas de lentitud y sobrecarga. Es frecuente ver mensajes de que la aplicación no ha podido abrirse debido a falta de memoria, así como encontrarse con que el aparato deja de responder durante varios segundos tras la apertura o cierre de una aplicación. Estos problemas se acrecentan cuando se trata de navegar por

Internet y cargar páginas, como las actuales, llenas de contenido multimedia principalmente flash.

El GPS del dispositivo N810 es especialmente conflictivo. En primer lugar el tiempo de sincronización es de varios minutos. Paseando por la calle suele ser imposible de sincronizar y para lograrlo hay que parar en algún lugar sin edificios altos alrededor y esperar unos minutos con el dispositivo en alto para que localice el número de satélites necesarios (siete). Además existen serios problemas de bloqueos del GPS. Cuando se entra en un edificio y se pierde la señal de los satélites, a veces hay que reiniciar el dispositivo para que vuelva a funcionar correctamente. Este tipo de problemas de sincronización son comunes a muchos dispositivos, pero se tratan de solventar al añadir a la información del GPS la información de posicionamiento por GSM o de forma manual, el denominado GPS asistido (A-GPS). De esta forma con menor número de satélites es posible obtener la posición. Para Maemo existe una versión Beta de A-GPS que, según la experiencia durante este proyecto, no soluciona los problemas de sincronización.
También muchas aplicaciones parecen dejarlo bloqueado, como el propio servicio de mapas incorporado. Por último cabe señalar que el nivel de uso de la batería es excesivo. Con el GPS activado tras un paseo de 3 o 4 horas la batería se agota.

3.2. Componentes Software

Maemo es básicamente un sistema GNU/Linux con escritorio GNOME preparado para dispositivos reducidos. Esta adaptación se consigue con una extensión de las librerías GTK+ denominada Hildon UI.

Por lo demás, casi todo el resto de componentes del *framework* de trabajo de Maemo ya están disponibles en un escritorio GNOME:

GnomeVFS: para la gestión de archivos.

Dbus: para facilitar la comunicación entre aplicaciones.

Pango,ATK y Cairo: para enriquecer la apariencia gráfica y tipográfica de las interfaces.

Gstreamer: para trabajar con flujos multimedia.

Tan sólo algunos elementos son totalmente propios de Maemo:

libConIC: para gestionar las conexiones a Internet

liblocation: para facilitar la programación de aplicaciones que usan GPS

Hildon: los mencionados nuevos widgets (en el sentido de componentes gráficos de una interfaz de usuario) adaptados a un terminal táctil de dimensiones reducidas.

Nokia estima que la plataforma es libre en un 90 % siendo el resto básicamente drivers propietarios de hardware. Por debajo de todo esto se encuentra un sistema Linux casi convencional compilado para el procesador ARM: kernel, utilidades de desarrollo GNU o shell de Bash compacto. Resaltar que en lugar de utilizar Xwindow o Xorg la versión 4 de Maemo utiliza el servidor gráfico Kdrive.

3.2.1. GTK+ y GNOME

GTK+ son las librerías gráficas que se programaron para desarrollar el famoso software de retoque fotográfico GIMP. Posteriormente el proyecto GNOME las tomó como base para construir su escritorio. Proporcionan los elementos gráficos más comunes en cualquier entorno gráfico: botones, scroll, formularios, etc. Están programadas en lenguaje C pero existen enlaces para utilizarlas desde prácticamente cualquier lenguaje algo difundido: Python, PHP, Java, Mono, etc.

3.2.2. Hildon

Hildon extiende GTK+ para dispositivos móviles. Fue desarrollado originalmente por Nokia para el escritorio Maemo, pero ahora se encuentra integrado dentro del proyecto GNOME. Principalmente añade un conjunto de elementos gráficos especialmente pensados para ser manejados desde interfaces táctiles reducidas. También incluye programas básicos como el propio escritorio, y un navegador de archivos y un sistema gestor de paquetes basado en los .deb de Debian GNU/Linux.

Debido a que el código de Hildon es libre, ha sido tomado por Ubuntu como escritorio de su versión "Ubuntu Mobile Internet Device Edition" que puede instalarse en dispositivos reducidos que funcionen bajo procesadores Intel. Esto puede proporcionar una mayor base de usuarios a las aplicaciones escritas para Maemo.

3.3. Documentación

La calidad en la documentación disponible para desarrollar en Maemo es muy variable. Por un lado se encuentran APIs completamente documentadas como son las de las librerías GTK+. En este caso hay artículos y tutoriales sobre casi todo lo que se necesite.

La calidad disminuye cuando se trata de los elementos propios de Hildon. En algunos casos la descripción de las funciones se omite y la documentación consiste simplemente en una enumeración de métodos.

Por último, los mayores problemas se encuentran a la hora de encontrar documentación propia de Hildon escrita para los programadores de Python. En estos casos hay que olvidarse prácticamente de la documentación oficial y recurrir a búsquedas en foros y listas de correo. La documentación más completa en este caso data de la versión 3 de Maemo.

3.4. Presente y futuro

Además de los cambios en hardware mencionados, el sistema operativo también ha evolucionado. Sin contar actualizaciones, han sido lanzadas 4 versiones de Maemo: 2005, 2006, 2007 y 2008. Los widgets que se desarrollan en este proyecto sólo son válidos para la versión 2008 puesto que el escritorio fue totalmente renovado entre la versión 2007 y 2008. Esta versión se incluyó específicamente con el dispositivo N810, pero debido a que el hardware es el mismo, rápidamente Nokia distribuyó un versión oficial para el dispositivo N800.

El futuro de la plataforma es algo incierto. Ya se prepara la nueva versión 2009 que incorpora algunas mejoras: soporte para los procesadores OMAP3, soporte para telefonía 3G, soporte para cámaras de alta definición, aceleración gráfica basada en hardware, etc. Sin embargo la llegada de nuevos dispositivos portátiles como los UMPC (PC ultra-móvil)y los actuales *smartphones* como iPhone han reducido el espacio disponible para una PDA sin teléfono incorporado y pese a esta nueva versión del software, no se vislumbra un futuro dispositivo de la serie N próximamente.[10]

Es importante para el futuro la adquisición por parte de Nokia de la empresa TrollTech, desarrolladora de las librerías libres Qt, "rivales" de GTK+ en el mundo Linux. Actualmente Nokia pretende hacer de Qt su librería estándar integrándola en todos sus teléfonos móviles para conseguir facilitar el desarrollo de aplicaciones en sus terminales. Esta noticia vino seguida del abandono de Nokia por parte de los principales desarrolladores de Hildon [21]. Así pues, el futuro de la plataforma Maemo, tanto a nivel hardware como software es incierto. Para lo que afecta puramente a este proyecto, si se portara la actual arquitectura de Maemo basada en GTK+ a otra basada en Qt, el esfuerzo de reimplementación sería muy alto.

3.4.1. Nivel de penetración, comunidad y soporte

La aspiración de Nokia no es disponer de un dispositivo para venta al gran público, sino que pretende construir una plataforma abierta de navegación para Internet. Nokia no proporciona cifras oficiales de venta, pero antes de la presentación del N810 cifraron en 300.000 el número de terminales vendidos de las series anteriores. Su público principal es el desarrollador que quiere tener Linux en su PDA, no el usuario final. [9]

La comunidad de Maemo se estructura entorno a dos páginas webs:

- maemo.org [2]
  Es la página oficial de Nokia para la comunidad de desarrollo. Contiene toda la documentación, Wikis, Bugzilla y un repositorio de aplicaciones.
- internettablettalk.com y tabletscene.com[11]
  Son dos páginas hermanas y complementarias a las de Maemo.org. No tienen relación directa con Nokia.

Ambas cuentan con su foro y aunque el nivel de actividad no es demasiado alto si existen varios post al día y es fácil recibir ayuda al poco tiempo de haber presentado la pregunta, especialmente en los foros de internettablettalk. La temática va desde cuestiones sólo interesantes para desarrolladores, hasta consultas sobre la utilización de algún software o función del terminal.

4. Desarrollando para la plataforma Maemo

En un proyecto en el que se trabaja con terminales poco maduros, es importante conocer las opciones de desarrollo que existen y probar hasta dar con el método más adecuado. En esta sección se exponen los lenguajes y herramientas que permiten escribir código para la plataforma Maemo, así como las distintas opciones que se han tomado durante la realización de este proyecto.

4.1. Eligiendo lenguaje

Las librerías gráficas GTK+ están escritas en y para el lenguaje C. A nivel gráfico los elementos Hildon que incorpora la interfaz de Maemo son una extensión de GTK+, y C es por tanto el principal lenguaje de programación en Maemo. Sin embargo la laboriosidad de desarrollar aplicaciones en C es elevada comparada con otras alternativas, como ahora se verá.

Por otra parte, este proyecto se desarrolla en un terminal pero requiere utilizar las tecnologías comunes de Internet, es decir, se va a trabajar a muy alto nivel y con lenguajes de intercambio y representación de información basados en etiquetas, principalmente XML. Por tanto sería deseable que el lenguaje de programación a utilizar proporcionara herramientas ágiles para trabajar con estas tecnologías: buena gestión de las cadenas de texto, parsers XML y gestión de conexiones HTTP. Aunque C contiene librerías que permiten trabajar con XML (*xmllib*) o HTTP, son complicadas de usar y el desarrollo puede hacerse muy lento.

Pese a esto, las ventajas de C están claras: proporciona velocidad de ejecución y un acceso total a las más sutiles opciones del sistema. Sin embargo en la actualidad no es tan importante la eficiencia como la sencillez de programación y la reducción de los tiempos de desarrollo. Por eso además de C, poco a poco se ha ido integrando el lenguaje de programación Python en la plataforma Maemo. En las primeras versiones era necesario añadir un gran número de paquetes a la versión base, pero ahora el lenguaje y el grueso de

sus librerías vienen integradas con el sistema operativo. La característica más conocida de Python es su facilidad de uso, y debido a que se ha convertido en uno de los lenguajes de programación más populares, se han desarrollado enlaces para utilizar Python con varias librerías gráficas como Qt o GTK+.

Por contra, las desventajas de Python son las ventajas de C. La integración entre los componentes propios del escritorio en Maemo y el lenguaje Python no es total. Esto es especialmente crítico en la serie de aplicaciones que se desarrollan en este proyecto, puesto que no son ventanas convencionales, sino elementos que se adjuntan al escritorio y que deben de estar soportados explícitamente. Además, la documentación disponible también es menor en Python que en C.

Como en todo sistema Linux, en Maemo existen además otros lenguajes que no son utilizados para el desarrollo de aplicaciones orientadas al usuario pero que cohesionan el sistema: BASH AWK o PERL, estos lenguajes no han sido tenidos en cuenta por ser más limitados.

Al final tras comprobar que era posible desarrollar widgets para escritorio en Python, se tomó la decisión de utilizar este lenguaje y descartar C. Por dos motivos: debido a la rapidez y la potencia para escribir código y sobre todo a su integración con los lenguajes de Internet. El parser de XML es extremadamente simple y además incluye librerías para procesar especificaciones concretas escritas en XML y de amplia difusión: ATOM, RSS, XHTML e incluso otras no XML como JSON. El protocolo HTTP también se encuentra bien soportado por la librería *httplib* incluida entre las librerías estándar de Python.

4.2. Widgets, Python y Maemo

Desarrollar un widget de escritorio es similar a desarrollar una aplicación de ventanas tradicional con la principal excepción de que no existen widgets en todos los sistemas de ventanas: el escritorio debe de soportar este tipo de

"ventanas incrustables". La potencia e importancia de los widgets ha variado en las distintas versiones de Maemo. En la versión actual forman parte del conjunto de elementos que pueden integrarse en el escritorio Hildon y se les conoce como plugins: "Task Navigator Plugin", "Status Plugin" y "Home Plugin". Siendo esto último lo que en este proyecto se está denominando como widget.

Mientras que en todas las versiones anteriores ha sido posible sin problemas adicionales, escribir aplicaciones gráficas utilizando Python, hasta la versión 2008 no hubo soporte, no oficial, para desarrollar estos Home Plugin en lenguajes distintos a C. El parche no oficial fue incorporado a la versión oficial en la versión 2008.1 (Diablo)

Desde el punto de vista del diseño de la aplicación, hay que tener en cuenta que la ventana principal del widget no debe de superar ciertas dimensiones para cohabitar con el resto de elementos que el usuario decida incorporar a su escritorio. Por lo demás el widget actúa como un contenedor similar a una ventana tradicional y por tanto se le pueden añadir los clásicos componentes de todo entorno gráfico: cajas, texto, listas, etc.

Uno de los principales problemas de trabajar utilizando widgets es el de la lentitud de los ciclos de desarrollo y prueba. Mientras que existe facilidad para desarrollar en un entorno y ejecutar y depurar automáticamente en Scratchbox o incluso en el propio dispositivo, el proceso para poder cargar el widget ha requerido siempre intervención manual. Además al cargar el widget desde el propio entorno gráfico, la salida de error y la salida por consola del programa no se mostraban haciendo imposible la depuración. Estos problemas fueron resueltos, si bien de forma poco elegante, como se explicará a lo largo de esta sección.

4.2.1. Otras opciones valoradas para desarrollar widgets

En lugar de desarrollar una aplicación específica para los terminales, en un principio se estuvo valorando portar una plataforma completa de widgets.

Existen varios servicios on-line que permiten crear widgets utilizando lenguajes propios muy simples. Estos widgets son elementos gráficos de tamaño reducido que se insertan dentro de otras páginas webs y que ofrecen una funcionalidad concreta, por ejemplo mostrar los correos no leídos o las noticias. Sin embargo algunos de estos servicios de webs van más allá y sirven un intérprete entre este sencillo lenguaje de programación y diversas plataformas además de la web: iPhone, Apple Dashboard, Windows Vista. Este es el caso de la web Netvibes.com. Esta aproximación, aún siendo la más compleja, es la que más cercanía tiene con el concepto actual de widget. Teniendo un interprete cualquiera podría escribir widgets para Maemo utilizando XML, CSS y algo similar a JavaScript. Por eso en las fases iniciales de este proyecto se contactó con Netvibes para explorar la posibilidad de portar a Maemo esa capa intermedia que transforma los widgets de Netvibes en widgets de otras plataformas. Esta solución podría poner a disposición del usuario de un dispositivo basado en Maemo, una enorme cantidad de widgets ya desarrollados. Sin embargo Netvibes respondió que aunque había algún plan para liberar el código de su motor de widgets (UWA) aún no sabían ni fechas ni qué porcentaje del código sería liberado. Por eso esta opción fue desestimada.

4.3. Historia: viajando a través de las opciones de desarrollo

Maemo es una plataforma aún inmadura en muchos aspectos. El potencial de una plataforma tan abierta es importante, sin embargo no se han pulido muchos detalles y esto provoca una sensación, tanto al usuario final como al desarrollador, de producto mal terminado debido a los cierres inesperados de las aplicaciones, bloqueos del sistema y documentación deficiente. En esta sección se narran las distintas decisiones han ido tomando a la hora de elegir el método de desarrollo. Los cambios vinieron motivados por el mayor conocimiento de la plataforma conforme se avanzó en el proyecto y por la necesidad de facilitar las tareas de escritura y prueba del software.

4.3.1. Scratchbox

Scratchbox es el entorno que proporcionó Nokia desde el primer momento para desarrollar aplicaciones para Maemo. Consiste en "un Linux dentro de otro Linux". Se toma un sistema Linux huésped (*host*) y dentro de él se crea un árbol de archivos que simula el de un sistema Linux completo pero que aprovecha del huésped recursos como el *kernel* y conectividad. Además lo complementa con otros elementos del escritorio y con herramientas de compilación cruzada. Para el entorno gráfico se utiliza Xephyr un software que permite ejecutar un servidor Xwindow dentro de otro. Así pues, en una segunda sesión Xwindow incrustada dentro de la principal, se dispone una sistema aislado cuyo funcionamiento no interfiere en el de su sistema huésped.

La principal utilidad de Scratchbox es la compilación cruzada. Los dispositivos de la plataforma Maemo utilizan procesadores de la familia ARM no compatibles con el juego de instrucciones x86 de Intel. Los dispositivos móviles pueden ser suficientemente potentes como para ejecutar aplicaciones complejas, pero el proceso de compilación puede requerir mucho más tiempo que si se compila desde un PC. Por eso una solución es compilar el software en el PC usando los compiladores para el microprocesador ARM. El problema es que este proceso de compilación cruzada no está exento de problemas de uso. Los desarrolladores de Scratchbox aseguran que usando esta plataforma se eliminan en gran parte los problemas de la compilación cruzada y se obtiene una gran ventaja: la posibilidad de desarrollar software para plataformas basadas en GNU/Linux antes de que el hardware se encuentre disponible.

En este proyecto no se han aprovechado las ventajas de la compilación cruzada de Scratchbox: Python es un lenguaje interpretado. Por contra si se han sufrido algunos de sus inconvenientes. Por ejemplo la necesidad de reiniciar el entorno gráfico cada cierto tiempo y lo extraño de su estructura (principalmente en la relación con el equipo *host*). También se tuvo que reinstalar todo el entorno varias veces debido a que dejó de funcionar y a que empezaron a surgir problemas de dependencias entre paquetes .deb continuamente.

Problemas de la puesta en marcha

En el momento de dar comienzo este proyecto, ya existía la librería necesaria para poder desarrollar Hildon Desktop Home Plugins en el lenguaje Python. Este código estaba disponible en uno de los SVN de Maemo, pero no estaba ni en los repositorios oficiales ni compilado, por lo que era necesario realizar el proceso de compilación e instalación en el dispositivo de forma manual. El problema fue que el código no podía compilarse de ninguna forma. Tras algunas semanas vagando por los foros, el propio desarrollador proporcionó un parche que solucionaba ese problema de compilación en el entorno Scratchbox y tras unos días de errores más, también en el dispositivo N800, justo dos de semanas antes de que liberaran el paquete oficial *libhildondesktop*.

Flujo de trabajo con widgets y Scratchbox

En la figura 4 se muestra el flujo de trabajo seguido en el desarrollo de los tres primeros widgets: .LRN, Flickr y Twitter.

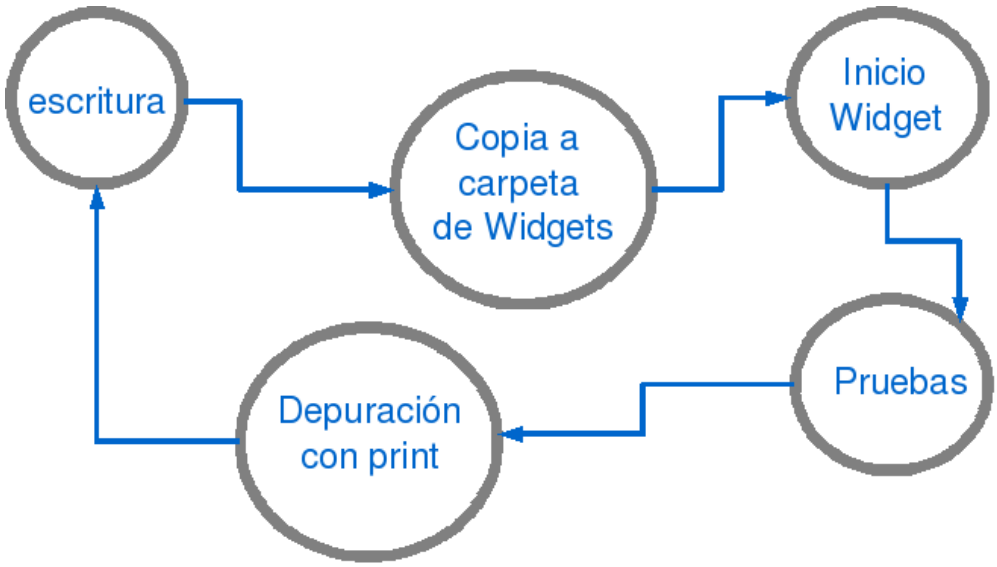

Figura 4: Flujo de trabajo con widgets y Scratchbox

La fase de escritura se llevo a cabo utilizando un editor funcionando en el sistema host. Tras completar alguna etapa y para hacer las pruebas se copiaba

el fichero a Scratchbox. Desde ahí se iniciaba el entorno gráfico y se probaba el widget cargándolo desde el menú en el escritorio de Maemo. De esta forma es fácil ver los mensajes de error en cada momento en la consola de Scratchbox, pero no es posible depurar con un debugger clásico. Tras corregir problemas, el proceso se repetía, siendo necesario en muchos casos reiniciar el entorno gráfico del sistema alojado para que se leyeran los cambios. Por último cada cierto tiempo se probaba el software en el dispositivo físico N800/N810

4.3.2. Desktop+Eclipse+Pluthon

Para abordar el desarrollo de los siguientes plugins se hacía necesario un cambio en el método de trabajo. Su mayor complejidad tanto a nivel de interfaz como de funcionalidades hacía difícil el desarrollo de la forma anterior. La estrategia a seguir fue la siguiente. La mayor parte del código que se desarrolla para Maemo 2008 es idéntico al código que se podría desarrollar en un escritorio GNOME. Por tanto en lugar de trabajar directamente en la plataforma Maemo se desarrolla el código en el PC, anotando los lugares en los que las librerías Hildon van a sustituir a las librerías GTK+ a la hora de ejecutar en el dispositivo móvil. También en lugar de desarrollar directamente sobre un widget de escritorio se desarrolla como un programa de ventana tradicional.

Cuando se ha desarrollado lo máximo posible en el PC llega la hora de pasar al dispositivo móvil. En esta etapa se sustituyen los componentes GTK+ necesarios por sus equivalentes Hildon. También, en algunos proyectos, éste es el momento de añadir el código relacionado con el GPS. En Linux existen varias formas de acceder al GPS que teóricamente deben de funcionar directamente en Maemo. Sin embargo en los foros se han reportado problemas y se recomienda utilizar las librerías propias *liblocation*. Por eso esta etapa de desarrollo se debe de realizar en el propio dispositivo. Por último, la tercera fase supone pasar de una aplicación en forma de ventana a una aplicación añadida al escritorio.

Para facilitar el desarrollo en el dispositivo existe un plugin para Eclipse denominado Pluthon. Este plugin permite depurar y ejecutar aplicaciones en

el propio dispositivo y está específicamente pensado para aplicaciones escritas en Python. Pluthon se apoya en el plugin Pydev para Eclipse que permite desarrollar en Python desde este IDE. Al ejecutar una aplicación se transfiere al dispositivo a través de SSH y se pone en marcha. Después incluso es posible controlar su ejecución con el debugger. La única pega es que no se soportan Hildon Desktop Home Plugins. Es por esto que primero se debió desarrollar en forma de aplicación tradicional y después pasar de aplicación tradicional a widget. Esta última etapa fue la que más problemas ocasionó.

De ventana a widget

En la arquitectura de Maemo un widget es una clase que hereda de *hildon-desktop.HomeItem* que a su vez hereda de *gtk.Bin*, al igual que las clases de ventana *gtk.window*. Sin embargo la cantidad de métodos y propiedades que se han implementado en los HomeItems es mucho menor que en las ventanas tradicionales. Por esto ciertas funciones de dibujado directo en pantalla a través de widgets GDK *gtk.gdk.Drawable* no eran posibles de la misma forma, teniendo que realizarse modificaciones al código.

Otro problema fue el de las transparencias. Las actuales librerías de GTK+ proporcionan transparencias utilizando las librería gráfica CAIRO. En principio según varios tutoriales y ejemplos de la red este tipo de dibujado en capa ALPHA estaba recién soportado en los HomeItems escritos en Python. En la práctica, tras muchas pruebas y problemas, en los foros de ayuda se llegó a la conclusión colectiva de que tan sólo se soportan transparencias en HomeItems con elementos del tipo *GTK.Label*. En el momento en el que intentan añadirse elementos de otro tipo, el escritorio de Hildon se reinicia. [12]

Sin embargo no han sido estos los problemas con más trastornos, sino la imposibilidad de automatizar el inicio y de depurar u obtener los mensajes de error del propio programa y del módulo encargado de cargar los widgets. Una aplicación puede arrancarse desde la consola y obtener así los mensajes de error propios y de sus librerías, sin embargo un widget no. Para solucionar ésto, es necesario ejecutar el siguiente comando en una sesión SSH:

```
#killall hildon-desktop;
run-standalone.sh /usr/bin/maemo-summoner /usr/bin/hildon-desktop.launch
```

Su resultado es destruir el escritorio actual e iniciar una nueva instancia del escritorio con todos sus módulos, incluido el encargado de cargar los widgets. De esta forma los mensajes de error de estos módulos pasan a la consola que ejecuta el comando y al cargar el widget es posible ver qué errores se provocan. Por desgracia tras una serie indeterminada, pero corta, de intentos de cargar un widget, el sistema colapsaba y era necesario reiniciar de nuevo el dispositivo, en este caso el N810.

4.3.3. Glade: editor de interfaces gráficas

Para conseguir un desarrollo de aplicaciones más rápido queda por resolver el asunto de la interfaz. Escribir el código necesario para colocar cajas de texto, etiquetas o listas es un proceso siempre similar pero tedioso. Por eso es común que existan editores de interfaces gráficos. En GTK+ se cuenta con Glade. En la figura 5 se muestra una figura del área de trabajo de Glade. En ella se disponen los distintos elementos gráficos que se ofrecen en la librería GTK+: cajas de texto, controles de scroll, tablas, etc. Simplemente arrastrando esos controles, sin escribir ninguna línea de código es posible diseñar la interfaz. Después a cada control se le asocia un nombre y, si se desea, una serie de eventos ante los que responder. Por último en el código fuente se toman los nombres de los objetos y los eventos y se asocian las funciones de *callback* correspondientes para llamar a esos eventos. Por ejemplo:

```
dic = {"search_clicked" : self.searchClicked}
self.gladeTree.signal_autoconnect(dic)
```

Como único problema se tiene de nuevo la dificultad de trabajar con widgets y no con ventanas tradicionales. En Glade el elemento principal del que cuelgan el resto de elementos es una ventana. El truco seguido para poder trabajar

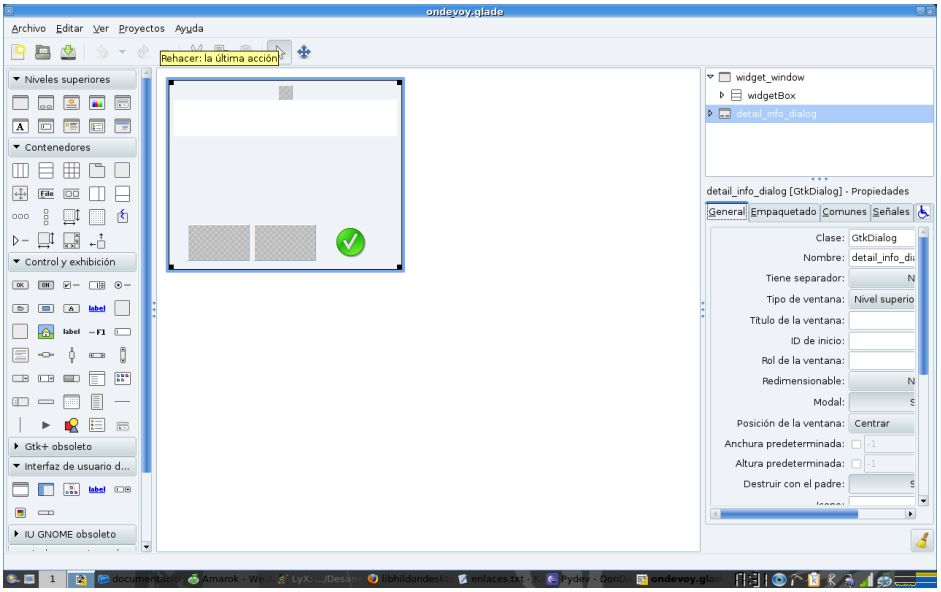

Figura 5: Diseñando Interfaces con Glade

con widgets ha sido incluir como único elemento hijo de la ventana principal una caja que contiene el resto de elementos y después, ya en el código fuente, se asigna el widget como nuevo elemento padre de esa caja:

```
self.vbox1.reparent(widget)
```

Glade almacena toda la descripción gráfica de la interfaz en un fichero XML que es el que hay que abrir para cargar todos los componentes:

```
self.gladefile = _thisDir + GLADEFILE
self.gladeTree = gtk.glade.XML(self.gladefile)
```

gladeTree será una estructura en árbol de todos los los elementos a los que es posible acceder de la forma padre.hijo

4.4. Resumen

Aparte del software en sí, una gran parte de la dificultad de este proyecto ha sido explorar el desarrollo en una plataforma no muy madura. A esto se

añade el extra de intentar desarrollar unos widgets complejos, lo que fuera de los desarrolladores de Maemo, apenas ha hecho más gente.

Es por esto, y por el uso del GPS del dispositivo N810, que la forma final de desarrollar ha terminado siendo muy distinta a la forma en la que comenzó. En el caso de un proyecto que por su naturaleza necesite estar escrito en lenguaje C y obtenerse el binario cruzado, es posible que la mejor opción sea otra (por ejemplo trabajar con Scratchbox). Por esto, y sin otros condicionantes externos, ante cada tipo de desarrollo para esta plataforma debe decidirse específicamente el entorno a elegir.

5. Desarrollo de widgets para Maemo

Como se ha contado en la introducción, el proyecto en sí comprende la programación de 5 widgets en nivel creciente de complejidad. En el desarrollo de cada widget se aprendieron cosas que se aplicaron en los siguientes. Por ejemplo ha ido evolucionando la forma de desarrollar: desde un simple editor gráfico a un IDE (Entorno de Desarrollo Integrado) como Eclipse. También el conocimiento del lenguaje Python y de las librerías GTK+ comenzó siendo nulo, con lo que muchas partes de los primeros widgets, podrían ser reescritas de otra forma seguramente más elegante aplicando los conocimientos adquiridos durante todo el proceso.

Los widgets se agrupan en 2 categorías, unos primeros que acceden a aplicaciones web como .LRN, Twitter y Flickr y otros que además hacen uso de la información geográfica. Su característica común es que pretenden hacer más rápido y útil el uso de varios servicios transformándolos en aplicaciones de escritorio.

5.1. Estructura común de todos los widgets en Maemo.

En Maemo un widget, o Hildon Desktop Home Plugin, escrito en Python requiere de un mínimo de dos archivos. Por un lado un archivo .desktop que contiene el nombre del widget y el nombre del archivo con la clase que hereda *hildondesktop.HomeItem.* Este segundo archivo contiene además de esta clase una función *hd_plugin_get_objects()* que se ejecuta primeramente y que devuelve un objeto del tipo *hildondesktop.HomeItem.* El archivo .desktop se coloca en /usr/share/applications/hildon-home/ mientras que el archivo .py con el código se coloca en /usr/lib/hildon-desktop.

5.2. Interactuando con .LRN

.LRN es un *Learning Management System* de código abierto. Su arquitectura se apoya en el *framework* OpenACS y todo el conjunto está programado en el lenguaje TCL. Como todo este tipo de software, internamente encontramos un núcleo principal y un potente sistema de módulos que pueden ser cargados y que aportan funcionalidades al sistema.

5.2.1. Objetivos

En sus inicios el objetivo de este widget era muy ambicioso, tanto que podía cubrir por si sólo el ámbito completo del proyecto. Se pretendían desarrollar diversos widgets que mostraran información procedente de .LRN de forma útil al usuario. Por ejemplo, mostrar los nuevos mensajes de los foros de las comunidades a las que el usuario perteneciera.

Para explorar la posibilidad de desarrollar este tipo de widgets para .LRN, como prototipo simple se pensó en mostrar un listado de usuarios conectados. Sin embargo, tras escribir este widget no se siguió trabajando con .LRN al no encontrarse, por error, una forma amigable de interoperar con aplicaciones externas: ningún tipo de salida XML ni ninguna API de servicios web. Debido a esto se tomaron dos aproximaciones: por un lado modificar el código de .LRN para que devolviera los datos con metainformación (XML) y por otro intentar analizar la salida HTML para obtener de ahí la información necesaria.

La funcionalidad de listar usuarios conectados en .LRN se encuentra en el paquete *whos-online*. Se eligió este paquete para empezar a trabajar por tener una estructura y una salida muy simples, así que es abordable sin tener demasiados conocimientos previos sobre .LRN

5.2.2. .LRN y sus plantillas para la capa de presentación

Un paquete de OpenACS se compone de varios archivos: por un lado el *index.tcl* que es el fichero que se lee primero en cada carpeta. Después para cada página tenemos tres ficheros distintos que podemos asociar a la estructura modelo-vista-controlador común en las aplicaciones web:

- Una página *.adp* que contiene HTML y un lenguaje de etiquetas propio y que permite hacer estas páginas dinámicas, insertando contenido desde el modelo de datos.
- Una página *.tcl* - Controla permisos y otros aspectos de la lógica del software. Proporciona los datos a la página .adp
- Una página *.xql* que contiene las sentencias SQL a ejecutar en la base de datos.

Para este proyecto la parte interesante se encuentra en la página *.adp*, tanto de cara a transformar el propio archivo .adp para obtener XML como para procesar el formato HTML. Si se explora un poco la relación entre los ficheros *.adp* y *.tcl* vemos que en el fichero .tcl se define una sección del tipo:

```
properties{ note_id:onevalue
submit_label:onevalue
target:onevalue
page_title:onevalue }
```

Las variables que se incluyen en ese bloque se pueden leer luego desde el fichero *.adp* metiéndolas entre *@variable@*. También se pueden usar plantillas maestras que definen una apariencia general. Como ejemplo simple de esto se muestra el código del fichero *.adp* del paquete *whos-online*.

```
<master>
<property name="title">@title;noquote@</property>
<property name="context">@context;noquote@</property>
<p>@summarize_categories@
<div style="font-size: 90 %"> @t1;noquote@ </div>
```

Por último gran parte del código de .LRN hace uso de los *listBuilders.* Utilizando en el fichero *.adp* el comando *<listtemplate name="nombre"> </listtemplate>* Se ejecuta un código que construye automáticamente tablas HTML. Esto ahorra mucho trabajo a los programadores, pero hace las cosas más complicadas si se quiere eliminar la salida HTML y sustituirla por XML.

5.2.3. Interpretando HTML

Como se ha dicho, HTML no está diseñado para proporcionar metainformación. Por tanto no se puede encontrar directamente un contenido determinado buscando en un fichero HTML. Para encontrar algo en concreto de forma automática hay que inspeccionar el archivo HTML de salida y encontrar pistas que indiquen dónde se encuentra la información que buscamos. Después hay que indicar esa información al parser HTML que se esté utilizando. En Python existe un parser especifico para SGML y HTML: *sgmllib.SGMLParser*. Su funcionamiento es similiar a un parser SAX-XML y es el siguiente:

1. Se crea una clase que herede de: *sgmllib.SGMLParser*
2. Se definen métodos del tipo *start_tag* y *end_tag* que son llamados cuando dentro del fichero a analizar se encuentra la etiqueta *tag*

Si se analiza la estructura del código HTML devuelto por *whos-online* en busca de parámetros que puedan indicar la posición, dentro del archivo, del nombre de los usuarios conectados, se ven tablas de la siguiente forma:

```
<td class="list-table" headers="online_users_name">
<a href="/dotlrn/community-member?user%5fid=71107">Sergio Galán</a>
</td>
```

Con lo que para obtener el nombre de usuario basta con crear una función que se llame cuando el parser encuentre la etiqueta *td* y que dentro se busque un atributo *headers* con valor online_users_*name*.

Si además se quiere almacenar el enlace a la página de usuario basta con llamar a una función que se active al encontrarse la etiqueta *<a>* y que almacene su atributo *href* siempre que se esté dentro de una etiqueta *<td header="online_users_name">*. Es decir, el mismo modo de funcionar que con un parser SAX de XML

Autenticación

A pesar de que, según la instalación por defecto de .LRN, para acceder a la información del paquete *whos-online*, no es necesario autenticarse, en previsión de escribir futuros widgets para acceder a información más restringida, se construyeron las funciones necesarias para autenticar al usuario.

El funcionamiento es idéntico al explicado en el caso de leer la salida de *whos-online* pero atendiendo a las peculiaridades el proceso de autenticación que sigue .LRN. Este proceso toma tres pasos:

1. Abrir *https://gradient.it.uc3m.es/register/*
2. Leer varios valores que aparecen en el código HTML como elementos ocultos de un formulario: *time*, *token_id* y *hash*
   Toman la forma:
   <input type="hidden" name="time" value="1200597621" >
   Por tanto hay que buscar el inicio de las etiquetas input y después diferenciarlas en función de su atributo *Name*
3. Enviar los anteriores valores junto con el nombre de usuario y la clave.

5.2.4. Modificando .LRN para obtener xml

La aproximación anterior presenta el problema de ser poco ortodoxa. HTML no está hecho para ser interpretado por otros programas. El parser anterior está ajustado al máximo a la salida HTML particular de un simple paquete y aún así pequeños cambios en ese HTML, que no tiene por qué ser constante entre diferentes versiones, invalidarían el código anterior. Si el HTML fuera más complejo y la información a recoger más amplia el problema se agravará aún más. Por eso ahora se toma la aproximación de modificar el código del módulo para obtener una salida simple y *machine-readable*. De esta forma el parser del lado del cliente es trivial y la complejidad pasa al lado del servidor.

Los cambios en el código del módulo modificaron los ficheros *.adp* y *.tcl* de la siguiente forma:

1. En *whos-online.tcl*: se añade el parámetro { {*xmlout "no"*} } justo antes de bloque *properties*, también se añade a este bloque (*xmlout:onevalue)* y se establece su valor en función de si se pasa en la URL(Uniform Resource Locator) o no: *set xmlout "$xmlout"*

2. En *whos-online.adp*: se toma la decisión de qué código mostrar en función del parámetro *xmlout,* y en el caso de que este parámetro aparezca, la salida será un simple fichero XML con el nombre de los usuarios conectados:

   <switch @xmlout@> <case value="no">

Así, al llamar a la web de la forma *http://mv-dotlrn:8020/shared/ whos-online?xmlout=yes* Se muestra una salida XML y al omitir el parámetro se muestra la salida tradicional. La salida XML en este caso se ha establecido de la siguente forma:

<?xml version="1.0" encoding="ISO-8859-1"?> <xml>

<user name= "Administrator" >

<url> /dotlrn/community-member?user%5fid=502 </url> </user>

</xml>

5.2.5. Estructura del cliente

El cliente HTML construido se basa en distintas clases que se encargan de analizar las distintas páginas que se presentan tal y como se ha explicado en el apartado 5.2.3. En este caso el proceso de autenticación previa no es necesario pero aún así se incluyó.

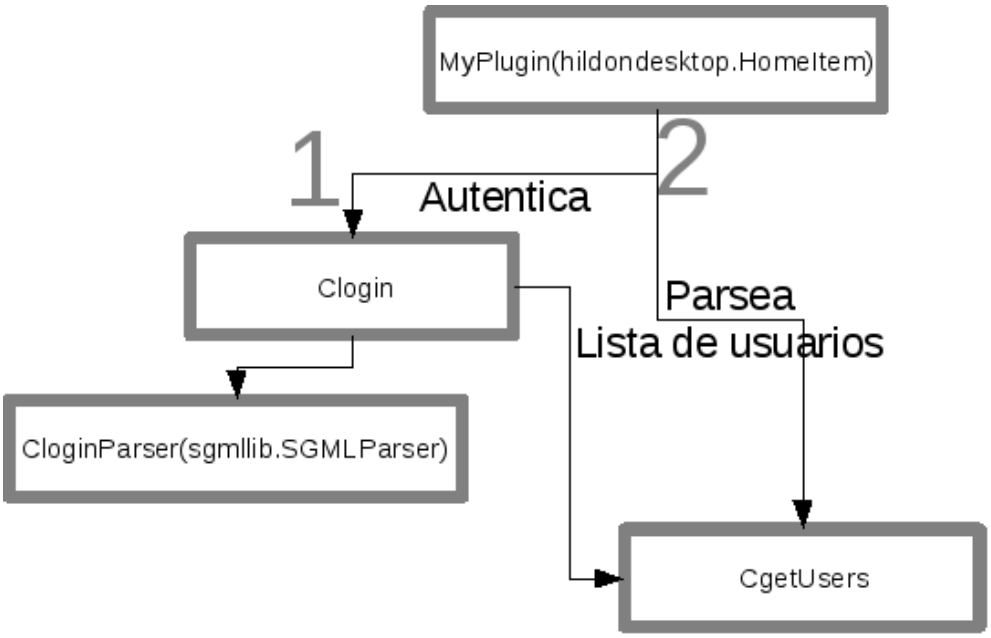

Figura 6: Estructura del widget para .LRN

Una vez que se ha conseguido la autenticación con la clase *Clogin*, las distintas páginas se van abriendo utilizando el manejador de *cookies* que incorpora Python en *urllib2.HTTPCookieProcessor* de forma que se mantenga la sesión. El esquema se muestra en la figura 6

5.2.6. Objetivos más ambiciosos

Este widget no deja de ser una prueba de concepto. Efectivamente se han demostrado dos formas de cómo es posible integrar .LRN con terceras aplicaciones, pero ¿Es factible?.

Si se trabaja con un paquete más complejo la salida es mucho más laboriosa de procesar. Habría que revisar el parseador de HTML tras cada nueva revisión de .LRN. El menor cambio en el código lo puede dejar inservible.

Generar el archivo XML es algo que se puede volver muy complejo en páginas con más datos. Además se ha visto como no sólo es necesario editar el archivo *.adp* de plantillas sino que también es necesario modificar el contenido del fichero *.tcl*. Esto supone de nuevo una tarea de edición continua del código de los paquetes ante cada nueva versión. Sin embargo meses después de la finalización de este widget, se descubrió que .LRN si que soporta la interoperabilidad con servicios externos utilizando el módulo de *webServices* de OpenACS. Lo que supone un error en la búsqueda de información para este proyecto y un incumplimiento de parte de los objetivos.

5.3. Fotografías con Flickr

Tras explorar la construcción de pequeñas aplicaciones que recogen información de sitios webs no preparados para compartir sus datos, el siguiente widget toma sus contenido de uno de los sitios pioneros en de la Web 2.0: Flickr. Este servicio constituyó uno de los primeros éxitos masivos tanto en la idea de construcción de una comunidad de usuarios como en el acceso a sus servicios desde aplicaciones externas. El servicio que ofrece Flickr es de almacenamiento y compartición de fotografías. Aunque actualmente otros servicios web como Facebook guardan una mayor cantidad de imágenes, Flickr se ha establecido como un servicio, no para compartir fotos personales, sino fotografías de calidad; tanto por parte de fotógrafos profesionales como de aficionados a la fotografía

5.3.1. Objetivos

Este widget también tiene una complejidad baja. Toma fotografías procedentes de Flickr, siguiendo distintos criterios establecidos por el usuario, y las muestra en el escritorio de Maemo, refrescándolas cada cierto tiempo. Estos criterios pueden ser:

- Fotografías favoritas
- Fotografías recomendadas por la comunidad Flickr.
- Fotografías públicas de un determinado usuario.

5.3.2. El API de Flickr

Flickr permite acceder a gran parte de sus funciones internas a través de tres formatos de servicios web: SOAP, XML-RPC y RestFul. Independientemente del formato, cada petición debe de contener información de autenticación

además del método que se pretende ejecutar. Se ofrecen métodos para cargar fotos, para añadirlas a los favoritos, para agruparlas, para clasificarlas o para realizar búsquedas avanzadas con parámetros que van desde el contenido de las fotografías hasta la posición. En definitiva, con su API se pueden crear aplicaciones externas que reemplacen totalmente a la interfaz web de Flickr.

API de Autenticación

Merece la pena detenerse en el proceso de autenticación puesto que es un proceso que se replica en otros muchos servicios web. Hay algunos cambios dependiendo de si la aplicación a desarrollar es una aplicación web o de escritorio, pero básicamente el proceso comprende los siguientes pasos:

1. Previamente hay que conseguir una clave API y asociarla con la aplicación. Esta clave hay que solicitarla a Flickr que a cambio pide información sobre el software. En este proceso también se asocia un secreto compartido.

2. Con esta información se construye una URL usando un *frob* dado por Flickr. En esta URL se le muestra al usuario una página pidiéndole autorización para permitir al software acceder su cuenta de Flickr.

3. Una vez que el usuario ha aceptado esta petición, la aplicación recibe un token que se pasará en cada llamada a los métodos de Flickr.

El proceso más detallado contempla los pasos de creación de la URL utilizando funciones de hash, sin embargo en este proyecto no ha hecho falta entrar en tanto nivel de detalle puesto que se ha hecho uso de una API para Flickr escrita en Python. De esta forma todo el proceso de bajo nivel para interpretar y transmitir mensajes con esta información se simplifica en 2 sentencias.

5.3.3. Binding para Flickr desde Python

Trabajar con servicios web es un proceso trabajoso. Utilizando un lenguaje de programación determinado hay que crear una serie de estructuras XML con parámetros dependientes de nuestra aplicación, enviarlos por HTTP y después interpretar la respuesta. Este proceso es muy mecánico y para muchos servicios web famosos existen enlaces que permiten trabajar de forma transparente con esos flujos de información XML que se pasan entre la aplicación y el sitio web. Usando estos enlaces el proceso es mucho más fácil. En este caso el paquete Flickrapi permite acceder a todas las funciones de la API de Flickr de forma muy integrada con el sistema de programación de Python. Por ejemplo, el mencionado proceso de autenticación se reduce a:

1. Función *flickr.get_token_part_one* devuelve el *frub* y un *token* temporal si la aplicación ha sido autorizada por el usuario o la URL a la que hay que acudir para este proceso.
2. Función *flickr.get_token_part_two* Almacena el *token* en cache y lo recuerda para futuros usos.

Después, a la hora de gestionar las peticiones y las respuestas todo se realiza trabajando directamente con funciones Python de la forma:

- Petición de fotos favoritas del usuario:
  self.flickr.favorites_getList(per_page='10')
- Respuesta: lista tradicional:
  self.ifavorites.photos[0].photo[num]

5.3.4. Menú de Configuración

Para la configuración de los Hildon Desktop Home Plugin, Maemo ofrece una opción especial accesible desde el mismo menú de carga de estos componentes. Para que una aplicación haga uso de este menú de opciones tiene que

implementar un manejador para la señal *settings*. En la figura 7 se muestra una captura de este menú.

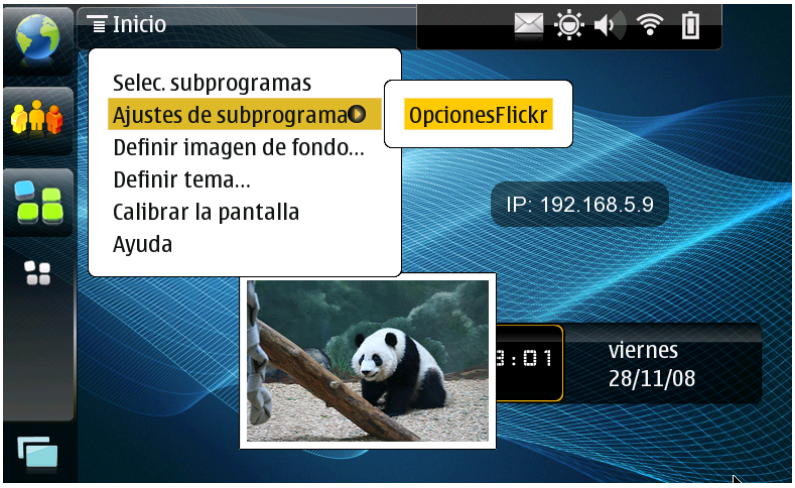

Figura 7: Menú de opciones de los widgets en Maemo

Una vez dentro se accede a una ventana en la que se puede configurar el tipo de fotos a recibir o el tiempo de actualización. Para almacenar esta información se recurre a otra clase estándar de Python incluida en Maemo: *ConfigParser*. El formato de archivo que genera esta librería es similar al de los archivos *.ini* de Microsoft Windows. El fichero queda dividido en secciones encabezadas por la etiqueta *[nombredeseccion]* y cuyo contenido son pares de valores de la forma: *refresh_time = 20*.

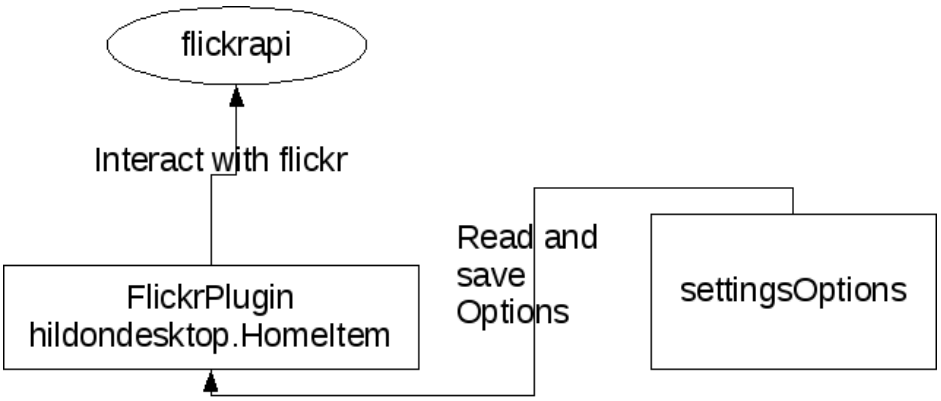

Figura 8: Estructura de Flickr Widget

El acceso y actualización de los datos es realizado a través de funciones set/get como: *configFile.get(section, option)*

5.3.5. Estructura final

La estructura del programa es reducida. Todo se encuentra en un único fichero con dos únicas clases. La encargada de leer las opciones de configuración y la encargada de mostrar la imagen en pantalla recibiendo la información necesaria desde Flickr con la ayuda de la librería Flickrapi.

5.4. Microblogging con Twitter

Twitter es uno de los servicios más simples de la Web 2.0. En el momento en el que apareció, muchos le aventuraban una corta vida, pero frente a estas predicciones, Twitter se ha establecido como un servicio muy popular. Su uso inicial es responder a la pregunta "¿Qué estás haciendo?" en menos de 150 caracteres. Cuando un usuario decide seguir desde su página de inicio las actualizaciones de estado de otro usuario se convierte en su *follower*. En la práctica el servicio se utiliza para recomendar enlaces, informar de actualizaciones de webs, de noticias, como chat asíncrono, e incluso los políticos lo utilizan para mantener contacto directo con los ciudadanos. La limitación de 150 caracteres se estableció para poder recibir actualizaciones y actualizar a través de mensajes SMS. Curiosamente, este límite, que se podría percibir como un problema, es justo lo que ha dotado a Twitter de su gran potencia como herramienta de comunicación instantánea.

5.4.1. Objetivos

Los objetivos de este widget son los siguientes:

- Recibir y mostrar la lista de actualizaciones de todos los usuarios de Twitter de los que el usuario de la aplicación es *follower*.
- Permitir actualizar el propio estado.
- Refrescar la información automáticamente.
- Permitir al usuario la configuración de parámetros como el número de actualizaciones o la frecuencia de refresco.

Es decir, se pretenden replicar las funciones básicas disponibles a través de la web de Twitter dejándolas siempre disponibles en el escritorio de Maemo.

5.4.2. El API de Twitter

Twitter también ofrece una API para interactuar con sus servicios. Debido a que el servicio es simple, su API también lo es. No incorpora un modelo de autenticación tan complejo como el de Flickr pero también permite replicar casi completamente sus funciones desde una aplicación externa. Su arquitectura es del tipo "RESTful" y acepta diferentes formatos de respuesta XML con direcciones del tipo *twitter.com/servicio.formato*. Por ejemplo para actualizar el estado: *http://twitter.com/statuses/update.format*

Funcionalidades

Es posible cambiar el perfil de usuario y la configuración, pero las opciones de uso más común son:

Autenticación Esquema básico de autenticación HTTP. Se realiza una petición al sitio de Twitter que responde con un mensaje HTTP 401 solicitando autenticación. Ahora el cliente responde con su nombre de usuario y contraseña codificados en Base64.

Timeline Conseguir la lista de actualizaciones de estado de los contactos.

Actualizar Estado Para enviar un mensaje que actualice el estado.

Búsquedas y tendencias Uno de los usos secundarios más interesantes que se da a la información que se encuentra en Twitter es el de visualizar de forma sencilla qué interesa al mundo en cada momento. El principal buscador de la red, Google, falla en este tipo de búsquedas en tiempo real, pero las tendencias de Twitter pueden mostrar segundo a segundo lo que más se mueve en la red, es decir, lo más comentado en cada momento por los usuarios de Twitter.

5.4.3. Construyendo un manejador para Twitter en Python

No existe ninguna librería que facilite el trabajo con Twitter desde Python. Por eso, y a diferencia de lo que ocurría con Flickr y la librería Flickrapi, hay que construir la interacción con Twitter directamente a través de las funciones de su API. Puesto que en este widget tan sólo se han hecho uso de las funciones de obtener la lista de actualizaciones y de actualizar el propio estado, sólo se han escrito métodos para "pythonizar" estas partes de la API de Twitter. Esto se realiza dentro de la clase *twitterapi* con los siguientes métodos:

- *getNames*: Transforma la estructura XML de respuesta de Twitter en una lista de la forma: *status[numero][nombre|imagen|texto|id]*
- *postStatus*: Envía una petición *Post* a Twitter con el nuevo mensaje de estado.

Para que estas peticiones sean tenidas en cuenta debe de existir una autenticación previa. Como se ha mencionado en el apartado referente a la API, Twitter no maneja estados ni incorpora un protocolo de autenticación complejo. Por contra se basa en el esquema de seguridad básico de HTTP. Por eso ante cada petición, Twitter solicita una clave y una contraseña. Mediante la librería *urllib* de Python se creó una clase que hereda de *FancyURLopener* y que permite realizar peticiones HTTP de forma que si el sitio solicita autenticación se proporcionarán las credenciales (user,password) devueltas por la función *prompt_user_passwd*

5.4.4. Interfaz Gráfica

La pantalla principal muestra a los *following* que han publicado recientemente un mensaje de estado. Junto con el nombre aparece la foto del usuario. Para ver el mensaje es necesario hacer click en el nombre. Difiere en esto de

la apariencia que toma la página web de Twitter, en la que se muestran los mensajes directamente. El motivo es reducir la cantidad de texto a mostrar para no ocupar toda la pantalla con un widget. La configuración se realiza, de nuevo, desde la ventana de configuración para los widgets que se encuentra en la parte superior del escritorio de Maemo. En la figura 9 se muestra una captura en funcionamiento. Los mensajes nuevos pueden ser mostrados automáticamente. También es posible agrupar varios mensajes de un mismo usuario, de esta forma al lado de su nombre aparecerá el número de mensajes disponibles que se irán mostrando cíclicamente al hacer click sobre ellos.

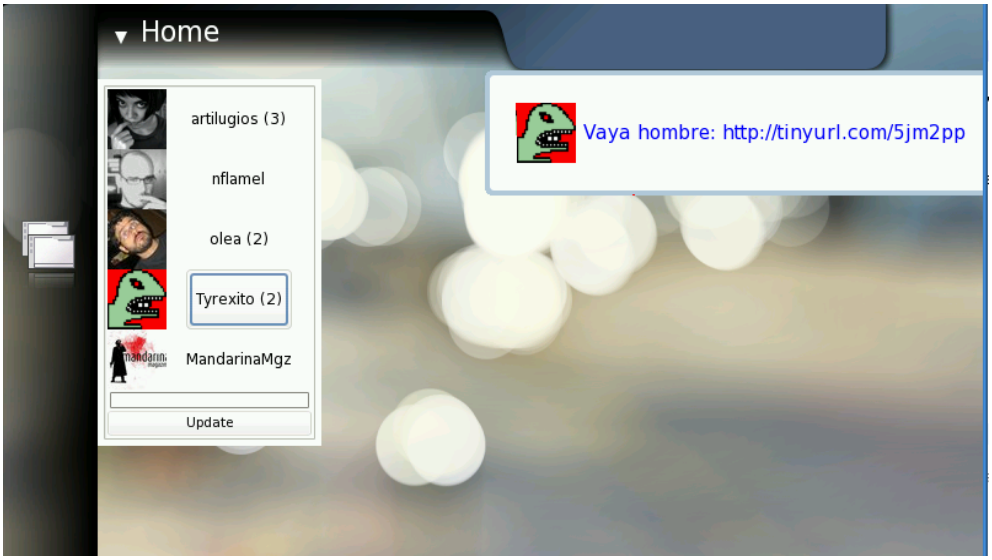

Figure 9: Plugin de Twitter funcionando

En este caso no se conocían demasiado bien las posibilidades de desarrollo y todos los elementos de la interfaz fueron codificados directamente sin hacer uso de ningún editor gráfico de interfaces.

Los problemas con las transparencias

Al igual que la forma de diseñar aplicaciones ha evolucionado, su aspecto gráfico también. Actualmente las ventanas de las aplicaciones suelen hacer uso de transparencias y disponer de formas más redondeadas que las diferencian de programas escritos algunos años antes. Basta ojear los widgets

embebidos en la versión 2007 de Maemo y en la versión 2008. Pero no es simplemente una cuestión estética. Por ejemplo, hay que pensar en encontrar en la actualidad una página web con diseño del año 1999, repleta de iconos animados y con mosaicos como fondo de texto. Es posible que tenga un contenido interesante pero al usuario de Internet actual, acostumbrado a un cierto tipo de estética, en un primer momento le será difícil tomarla como una web profesional y digna de atención. Esto mismo sucede con las aplicaciones y por eso es importante dotarlas de un buen aspecto.

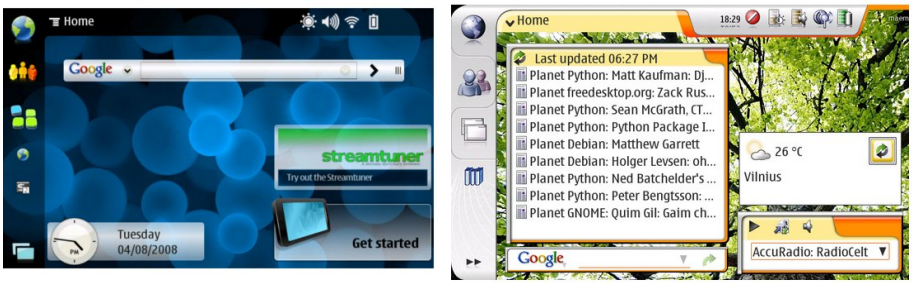

Figura 10: Transparencias en escritorios de Maemo 2006 y 2008.

Las transparencias y el redondeado en las ventanas de GTK+ se consiguen a través de la librería de dibujado "Cairo". En la red pueden encontrarse ejemplos de código en C y Python que dibujan un *gtk.Label* translúcido con un texto en él. Sin problemas se pudo escribir un widget similar, pero cuando se activaba la transparencia el escritorio completo se reiniciaba. Tras varios días buscando errores y cambiando configuraciones se llegó a la conclusión (en un hilo de discusión de internettablettalk.com) de que simplemente cambiando la etiqueta *label* por un *gtk.Button* el sistema colapsaba. Debido a que en los "Home Plugins" escritos en C este problema no se daba, se concluyó que era problema de un bug del binding de Python para hildondesktop, bug que a día de hoy no ha sido resuelto. [12]

5.4.5. Configuración

La configuración se almacena en un fichero *.twitterPlugin* que se crea dentro del directorio */home/user* de Maemo. Su estructura es la creada por la librería *PythonConfigParser* y que ya se ha comentado. En este caso las opciones son:

- Tiempo de refresco.
- Número de actualizaciones a mostrar.
- Tiempo que se muestra cada mensaje en la burbuja.
- Agrupar varias actualizaciones de un mismo usuario.

5.4.6. Estructura final

En la figura 11 se esquematiza la estructura de la aplicación. Además de las clases necesarias para manejar la interacción con Twitter y de la clase *TwitterPlugin* que se encarga de gestionar la Interfaz gráfica, la clase *settingsOptions* gestiona todo lo relativo al almacenamiento y recuperación de los datos de configuración. Durante la ejecución del programa estos datos están almacenados dentro del diccionario *config* compuesto por elementos con el mismo nombre que los disponibles en el archivo de texto de configuración.

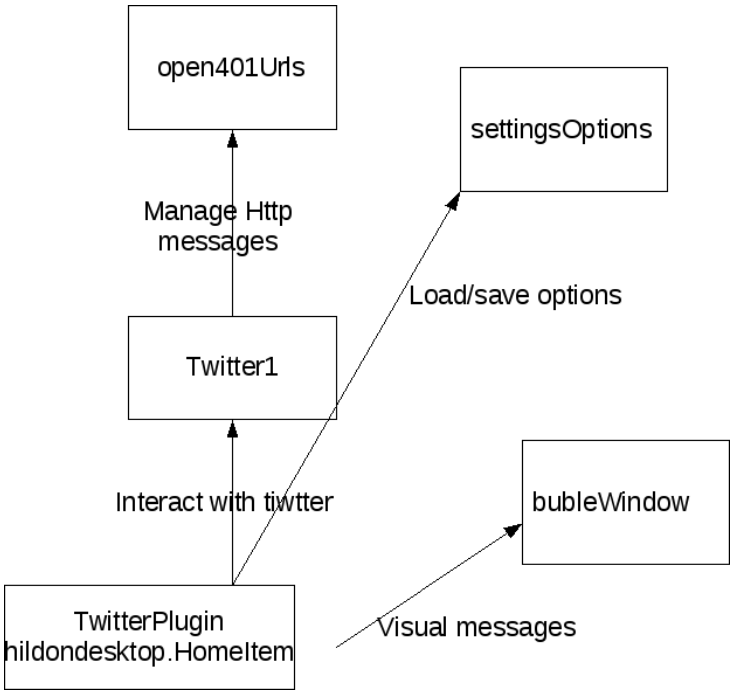

Figura 11: Estructura del widget para Twitter

5.5. getPan(): calendario geoposicionado

Éste es el primero de los dos widgets desarrollados que hace uso del GPS integrado en los Nokia N810. En él, se pretende jugar con los calendarios tradicionales y la idea del tiempo y el espacio. Existe la costumbre de utilizar agendas que recuerdan en qué ventana temporal hay que realizar una acción determinada. Con este widget, se propone que sea el espacio el que se asocie a una acción. Esta idea puede explorarse de diversas formas y aquí se limita a la organización personal. En esta aplicación además de contar con el GPS se ha hecho uso de Google Maps para obtener mapas de las zonas en las que se posiciona un evento determinado y también de herramientas de *geocoding* para obtener la posición GPS que se asocia a una dirección del tipo "Nombre de Calle + Ciudad"

5.5.1. Objetivos

Bajo el nombre de getPan() el objetivo planteado fue diseñar un calendario de tareas asociadas a lugares de forma que:

- Se pudieran agregar nuevas tareas asociadas a cualquier lugar.
- Una vez agregadas las tareas no hiciera falta conexión a Internet.
- Se implementara algún tipo de notificación para el momento en el que la tarea se active.
- Fuera manejable directamente con los dedos sin utilizar el *stencil*.

5.5.2. El Api de Google Static Maps

Google lanzó su servicio de Google Maps en febrero 2005 y su novedosa interfaz de uso, la calidad de las fotografías, y la API para crear aplicaciones

basadas en estos mapas, rápidamente lo posicionaron como uno de los servicios más utilizados de Internet. Además de ofrecer la interfaz completa en JavaScript, también se ofrece acceso directo a las imágenes utilizando una API especial denominada Google Static Maps. De esta forma se pueden recuperar imágenes individuales de tamaños menores a 800 pixels de mapas de cualquier zona del mundo. Esta opción era la más apropiada en este caso, puesto que la interfaz web JavaScript no puede ser empotrada directamente en el software de Maemo.

Las opciones de recuperación que Google Static Map ofrece son:

- Nivel de Zoom: Entre 1 y 16 siendo 1 el nivel más lejano.
- Tamaño de la imagen.
- Longitud/Latitud.
- Marcadores: Para colocar "chinchetas" y notas en el mapa.

Y la forma de acceder a ellos es simplemente mediante URLs del tipo: *http://maps.google.com/staticmap? center=40.714728,-73.998672 &zoom=12& size=400x400 &key=MAPS_API_KEY* . Se puede ver que se solicita una *API-key* para cada petición así como los parámetros de tamaño, zoom y centro.

5.5.3. *Geocoding* y *geopy*

Geocoding es el proceso de asociar coordenadas geográficas (latitud,longitud) a otro tipo de datos geográficos(calles, códigos postales, etc.) Es decir, se dispone del nombre de una calle y quieren averiguarse sus coordenadas. En este proyecto no se ha necesitado profundizar en cómo funciona el *geocoding,* sino que se ha hecho uso de algunos servicios de *geocoding* disponibles *online* por parte de empresas como Yahoo o Google. Pero la tarea es aún más sencilla porque para Python existe la librería *geopy* que permite hacer uso de estas herramientas directamente desde Python. *Geopy* tiene implementados varios

servicios, aunque en este proyecto se ha usado el de Google. Realizar búsquedas con *geopy* no lleva más de dos líneas: una para seleccionar servicio y otra para buscar. El resultado es una lista de las localizaciones que coinciden con los criterios de búsqueda, junto con sus coordenadas en longitud y latitud.

Geopy además también ofrece funciones para calcular la distancia entre dos puntos expresados en longitud/latitud: la distancia geográfica. Este cálculo no es trivial y existen distintas aproximaciones en función del grado de precisión que se necesite. Para puntos cercanos es posible tomar un modelo de tierra plana, pero para mayor precisión hay que recurrir al modelo de tierra esférica o elipsoidal. Aquí se usa el modelo elipsoidal, basado en las fórmulas de Vicenty, y que proporciona una precisión de 0.5 mm. Éstas fórmulas utilizan diversos modelos de elipses para realizar los cálculos, siendo WGS 84 el modelo más utilizado. [13]

5.5.4. Acceso al GPS

Para acceder al GPS Maemo cuenta con las librería *liblocation*. Para utilizar esta librería directamente con estos widgets escritos en Python se encontró una versión casi completa de un binding a *liblocation.py* [15] que tras un parcheo al código funcionó correctamente.

Para acceder al GPS del N800 usando *liblocation* hay que:

1. Inicializar el dispositivo.
2. Obtener un objeto de control del GPS.
3. Iniciar el objeto de control (si es que se puede tener el control).

Para leer los datos que se ofrecen hay que llamar al método *struct()* que devolverá una estructura el estado del GPS y una subestructura *LocationGPSDeviceFix* que contiene la información del GPS: latitud, longitud, velocidad, altitud...

Estos procesos se ejecutan en un hilo aparte y el programa principal cada cierto tiempo comprueba la información que el GPS ofrece. Antes de actuar es importante verificar que los datos son frescos comprobando el estado del dispositivo (que esté sincronizado) lo que se hace a través de la variable de *status*.

5.5.5. Interfaz y opciones de usuarios

El flujo de uso que se ha diseñado para la aplicación es el siguiente:

1. Añadir un lugar a la lista de lugares favoritos
2. Añadir una tarea asociada a un lugar favorito
3. La tarea aparece en la lista de tareas pendientes, esperando a cumplir la condición de cercanía respecto al lugar.
4. Cuando se cumple la condición se activa con el aviso predefinido y cuando el usuario acepta el aviso la tarea se desactiva.

En la ventana del widget aparece un listado de tareas pendientes. Al hacer doble click sobre la tarea se puede ver la descripción y un mapa de la zona. También están los botones de añadir y eliminar tarea así como el de gestionar los lugares favoritos. En la parte superior está el botón de activación del GPS que permanece en color rojo con el GPS desactivado, parpadea durante la sincronización y se colorea de verde cuando está en funcionamiento. En este último estado también se muestra la distancia restante para llegar a la tarea seleccionada en la lista. Para más información sobre la interfaz de usuario consultar el manual en el apéndice A.4.

5.5.6. Estructura general

La estructura de este widget es mucho más compleja que la de los anteriores. En primer lugar su GUI (Interfaz Gráfica de Usuario) se ha diseñado con

Glade. Como se ha visto en la sección 4.3.3 Glade ofrece como salida un archivo XML en el que se describe la interfaz y las señales de *callback* que se generan, de esta forma la programación de la GUI es una tarea mucho más liviana. Los archivos que componen getPan() están estructurados en base a su área de utilización y cada archivo puede contener varias clases. El esquema se aprecia en la figura 12 y los principales archivos son:

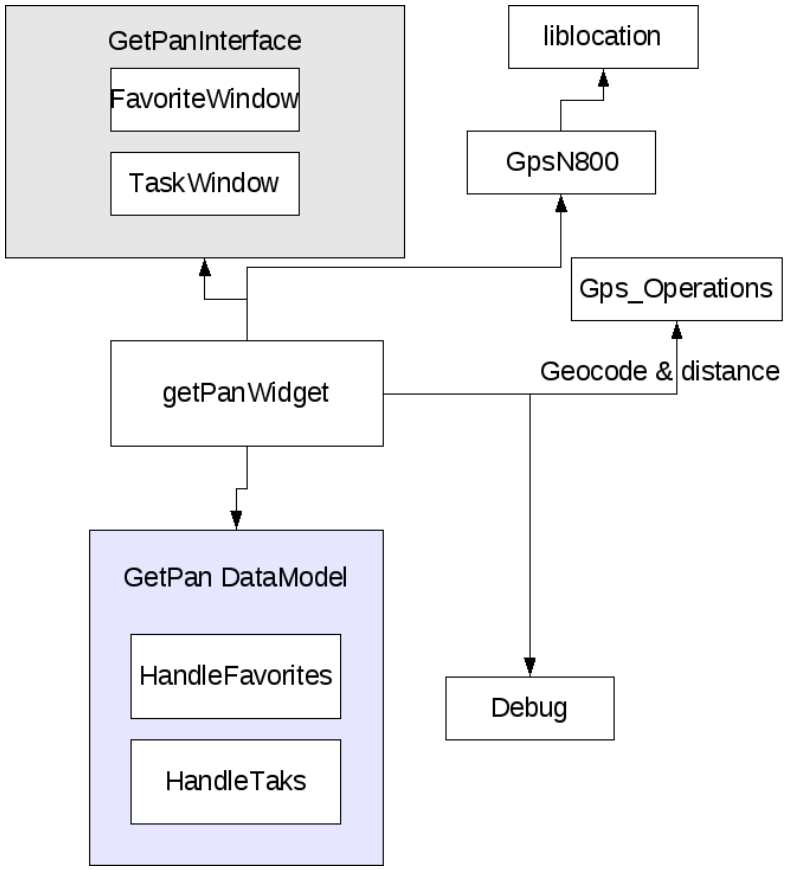

Figura 12: Estructura de getPan()

- GoogleMaps: *googleStaticMaps.py* Contiene el código necesario para crear URLS de Google Static Maps a partir de la latitud y la longitud.
- Geocoding: *geocode_operations.py* simplemente obtiene longitud y latitud para una URL y calcula la distancia entre dos puntos. Básicamente

es una función puente para acceder a las funciones de la librería *geopy*.

- Modelo de datos: *getPanDataModel.py* Es necesario manipular y almacenar los eventos y los lugares favoritos. Para el almacenamiento se vuelve a usar la librería *ConfigParser*.

- GPS: *gps_n800* Contiene las funciones necesarias para iniciar el GPS, chequear la distancia respecto a cada objetivo y detener el GPS. Depende totalmente de la librería *liblocation*.

- Interfaz: *getPanWidget.py* es la clase principal y contiene todas las clases encargadas de la representación de la interfaz gráfica del widget así como de la consulta periódica de la información del GPS. Por su parte *getPanInterface.py* es la encargada de representar las ventanas de añadir tareas y lugares.

5.6. Recomendación de servicios: DónDe

A la hora de salir por una ciudad la oferta de ocio que se ofrece puede ser enorme: montones de bares, restaurantes y pubs. El problema suele ser elegir qué sitios merecen la pena y, aún mas difícil: qué sitios merecen la pena en el entorno el que uno se encuentra. Para intentar ayudar en esta tarea, este último widget combina la información geográfica con fuentes de datos obtenidas de aplicaciones web. Se mezclará la información geográfica del GPS con la información de servicios recomendados del servicio web 11870.com. De esta forma se obtiene una lista de servicios acordes con unos criterios de búsqueda y cercanos al lugar en el que el usuario se encuentra. Este widget ha sido bautizado como "DónDe".

5.6.1. Objetivos

Los objetivos previstos en el subproyecto DónDe son los siguientes:

- Presentar al usuario una lista de lugares de acuerdo a un criterio de búsqueda y de un lugar.
- Actualizar la información automáticamente con el cambio de posición.
- Obtener información de determinados lugares bajo demanda del usuario.
- Disponer de una interfaz de uso fluida y sin esperas.

5.6.2. 11870.com

11870.com se define como un tarjetero virtual. Es un servicio web que permite almacenar lugares de una ciudad (teatros, cines, bares, pubs, restaurantes, etc.) asignarles un comentario y adjuntar fotos y vídeos. De esta forma un usuario puede almacenar sitios que ha visitado para volver (o no) a ellos.

También permite descubrir nuevos lugares interesantes efectuando búsquedas como "Bares en Malasaña" o "Cenar en París" y posteriormente leyendo las críticas de los lugares que se ofrecen como resultado de la búsqueda. En definitiva se podría definir a 11870.com como unas páginas amarillas 2.0.

API de 11870.com

11870.com cuenta con una API para acceder sus servicios [22]. Es extensa y permite realizar un gran número de operaciones desde las aplicaciones que la usen, entre ellas destacar:

Autenticación: sólo necesaria para aplicaciones que modifican el propio perfil del usuario añadiendo lugares o contactos. Utiliza un esquema similar al visto con Flickr:

1. Se obtiene un token temporal
2. Con el token temporal y los permisos a conceder se construye una URL que se abre al usuario solicitando confirmación
3. Se obtiene un token permanente.
4. Este token permanente se usa en las peticiones HTTP añadiendo una cabecera descrita por el algoritmo WSSE (Web Services Security) [27].

Búsqueda: la búsqueda se realiza anexando a una dirección base *http://11870.com/api/v1/search* una serie de parámetros, como por ejemplo:

q Para una cadena de texto.

lat,lon,radius Para la búsqueda en un entorno.

l Dirección que se convertirá a coordenadas automáticamente con un servicio de *geocoding.*

Resultados de la búsqueda: se ofrecen en formato ATOM/XML con diversas extensiones como:

OpenSearch: extensiones para mostrar datos relativos a la propia búsqueda, esto es: número de resultados totales, página actual, resultados por página, etc. Además cada servicio que se entrega como resultado se mapea también según el formato ATOM/XML con otras extensiones:

geoRSS: para etiquetar los datos geográficos.

oss: extensión propia para información de Número de teléfono, URL del servicio, país. etc.

Gestión: del propio perfil, de los contactos, de los servicios etc. En la actual versión de DónDe no se modifica este tipo de información.

Construyendo el enlace Python para el API de 11870

No existe ningún código que facilite el acceso a esta API desde Python. Por tanto ha tenido que escribirse todo el código para manejar las peticiones y respuestas desde 11870.com. Aunque también se ha escrito parte de lo imprescindible para la autenticación de los usuarios, no se ha probado debido a que no es necesario este proceso para realizar búsquedas. Así, la mayor parte del código se dedica a facilitar el análisis de los resultados y a acceder a los datos relevantes para la aplicación. Por tanto este código no puede considerarse una librería genérica para el servicio de 11870.com usando Python sino un código específico para este widget en concreto. Los métodos usados se encuentran dentro de la clase *base11870*.

Para buscar: el método *search* acepta una cadena de texto y un parámetro *mfigure* que es un diccionario de tres valores para una búsqueda del tipo "longitud,latitud,radio" o cuatro para una búsqueda en superficie cuadrada de la forma "latitud mínima/máxima, longitud mínima/máxima".

Procesado: por suerte Python cuenta con un analizador muy potente para archivos XML/ATOM: *Feedparser*. Durante el desarrollo con este parser el funcionamiento en las pruebas desde el PC era correcto, pero

inexplicablemente no reconocía etiquetas ajenas a las definidas propiamente en ATOM al usarlo en el N810. Hubo que insertar ciertas líneas de código -sin explicación aparente- para que el software funcionase correctamente. Por eso la versión de *Feedparser* que se utiliza viene incluida en el propio código de DónDe en lugar de utilizar la librería estándar. El procesado que realiza *Feedparser* permite acceder a todo el árbol ATOM/XML usando diccionarios, lo que es muy cómodo y apropiado para Python: *d['feed']['title']*. Pese a esto en algunos puntos hubo que afinar algo más a la hora de obtener cierta información por ejemplo:

Imágenes: en cada entrada, la etiqueta *links* contiene las URLs de distintas imágenes asociadas a un servicio de 11870.com identificadas bajo el atributo
rel='media'. Cada imagen contiene un prefijo que indica el tamaño y por defecto en la búsqueda todas las imágenes vienen dadas por el prefijo 'ps_' que define imágenes de tamaño pequeño. Para poder obtener las imágenes con un tamaño mas útil hay que sustituir el prefijo ps_ por pl_

Enlace al feed del servicio: hay que sustituir en una URL dada, como antes, por el atributo *rel='alternate'* de la etiqueta *links* el valor */pro/* por */feeds/pro*.

Comentarios: vienen ya marcados con etiquetas HTML que hay que eliminar, puesto que las cajas de texto de GTK+ no soportan HTML. Para esto se tomó una función ya escrita por "Fredrik Lundh" que se ha incluido con el nombre: *strip_html* dentro del archivo *api11870.py* [28].

Heredando la clase *base11870* está la clase *gtk11870* pensada para autenticar al usuario mostrándole los mensajes necesarios en un entorno GTK+, pero actualmente no se hace uso de esta funcionalidad.

Un ejemplo de búsqueda usando esta API sería:

```
self.my11870=api11870.gtk11870()
d={'lon':lon, 'lat':lat, 'radius':self.dist}
file1 = self.my11870.search('restaurante vegetariano',d)
search_results = self.my11870.parseSearch(file1)
entries = search_results.entries
```

5.6.3. Interfaz

Se plantearon dos interfaces para el widget. Una primera en la que simplemente se toma la lista de resultados devueltos, se muestra y posteriormente se accede a la información del lugar y de dónde está localizado.
La otra opción es mostrar los resultados de la búsqueda ubicados en un mapa del entorno. Esta segunda opción es mas intuitiva y útil si se consiguen solventar los problemas de:

- Acumulación de varios elementos en cercanos.
- Movimiento del Mapa.

También conlleva una dificultad técnica mayor, pero se tomó ésta segunda aproximación puesto que permite obtener una aplicación mas visual y rápida de usar.

Nuevos elementos gráficos

Para este widget se han desarrollado elementos gráficos de GTK+ personalizados partiendo de elementos de muy bajo nivel. Objetos como botones o listas se encuentran en los niveles inferiores de la jerarquía de clases de GTK, pero si se desean construir objetos gráficos mas ricos hay que ascender y programar directamente la gestión de los eventos de usuario, y hasta el momento de refresco de las imágenes.
Usando éste tipo de técnicas se ha escrito un objeto gráfico para generar transiciones dinámicas entre imágenes, un efecto de desplazamiento como el

que puso de moda el iPhone de Apple, si bien algo más rudimentario. Este efecto se aplica en DónDe en dos lugares diferentes: para navegar por las imágenes disponibles de un servicio y para realizar la transición entre mapas que se explica en el apartado 5.6.5

Existen varias formas para desarrollar este tipo de elementos gráficos a medida. En este caso la elegida ha sido partir de los elementos gráficos *gtk.EventBox* y *gtk.DrawingArea* que se implementan dentro de la clase *scrollImageView*. Con *gtk.DrawingArea* es posible dibujar directamente en pantalla utilizando primitivas básicas y controlar el momento de refresco y con *gtk.EventBox* se pueden capturar eventos en elementos no preparados por defecto para ello. Para realizar el scroll hay dos fases:

1. Detectar el movimiento del dedo "pasando" la imagen. Para esto se captura el punto en el que empieza el movimiento y se abre una ventana de tiempo en la que se espera capturar de nuevo un "click" en otro punto suficientemente alejado en el eje x (desplazamiento horizontal) o en el eje y (desplazamiento vertical).

2. Una vez detectado el sentido del desplazamiento hay que realizar una transición entre las dos imágenes. Para entender como se hace esto, primero hay que saber que en Xwindow cuando toca redibujar un elemento de pantalla se lanza el evento *expose* y para determinar cuándo dibujarlo, lo mejor es decir a XWindow que invalide un área de la pantalla. Una vez explicado esto, el proceso es como sigue:

    a) Se toma la imagen actual y la imagen a la que se va a llegar y se temporiza un desplazamiento por el cual cada X milisegundos la imagen futura "roba" Y pixels a la imagen actual.

    b) Se invalida el área de la imagen (con la función *invalidate_rect*) para que se redibuje cuando se pueda.

    c) Se itera hasta que la nueva imagen ha "robado" todo el espacio a la anterior.

Para el paso de 1 a 2 el mecanismo es el siguiente: Se define una función de *callback* que será llamada con un parámetro que depende del sentido del movimiento. La función de *callback* debe de interpretar ese parámetro para colocar la imagen hacia la que se dirige la transición haciendo uso de la función de *setNextImageGDK* de la propia clase.

Para el caso del desplazamiento entre mapas, el funcionamiento es idéntico.

5.6.4. Integrando las búsquedas de 11870.com con GoogleMaps

11870.com en su web ofrece un mapa de Google Maps con la localización del servicio. Sin embargo esto no es utilizable por esta aplicación de escritorio: los mapas que se muestran utilizan la versión web de Google Maps y esto requiere de un navegador con soporte AJAX para funcionar, algo que no puede integrarse en una ventana de Maemo. Por tanto para poder ofrecer el resultado de la búsqueda integrado en la aplicación hay que obtener imágenes estáticas. Para obtener una interfaz en la que aparece la posición del usuario y los servicios que se corresponden con la búsqueda representados en el mismo mapa, hay combinar manualmente los resultados de una búsqueda en 11870.com con un mapa estático de Google. Así pues se construirá una URL Google Static Maps que ya devolverá una imagen en la que aparezcan representados los servicios. Este proceso se esquematiza como sigue:

1. Tomar posición actual, tamaño en pixels del mapa resultante, nivel de zoom óptimo y realizar una búsqueda en 11870.com de acuerdo al texto introducido por el usuario. La búsqueda estará geográficamente centrada en la posición actual del usuario, y tendrá como límites los que se correspondan con el tamaño de la imagen. Para esto hay que convertir el tamaño de la imagen de pixels a kilómetros.

2. Analizar los resultados de la búsqueda y generar una lista de con los servicios disponibles y su posición geográfica.

3. Generar una consulta usando la API de Google Static Maps de forma que se obtenga un mapa centrado en la posición actual del usuario y en el que se encuentren marcas en la posición que corresponde a cada servicio. Ahora falta convertir esa imagen estática en una imagen con la que se pueda interactuar.

4. Asociar cada región de la imagen en la que hay una marca con un lugar. Para ello hay que convertir de longitud/latitud a pixels. Así, ahora cada resultado de la búsqueda tiene asociado un par longitud/latitud en el espacio físico y además un par x/y en la imagen. De esta forma cuando el usuario toca una región de la imagen es posible identificar qué servicio quiere observar.

En este proceso hay que realizar varias veces una conversión entre pixels y longitud/latitud y el código para realizar esto fue modificado a partir de un programa encontrado en la red escrito en JavaScript [14] y portado a Python.

5.6.5. Optimizaciones

Cuando el usuario activa un evento está acostumbrado a respuestas mas o menos rápidas: es lo esperable en una aplicación de escritorio. Las aplicaciones web han tenido que simular esto con peticiones asíncronas de información, que les permite estar preparadas para ofrecer al usuario lo que va a pedir después. Por eso la interfaz web de Google Maps no se limita a cargar el mapa que el usuario está viendo, sino que, sin que éste lo perciba, efectúa múltiples descargas de los mapas de las zonas de alrededor. Algo similar ha de realizarse aquí puesto que no es factible que cada vez que el usuario desplace su mapa, tenga que realizarse una petición a 11870.com, otra a Google Maps y componer la imagen antes de que producir la transición. Para esto se trabaja con hilos parealelos que se encargan de realizar peticiones en *background* de forma que descarguan no sólo el mapa central sino los que se encuentran alrededor. Señalar que la opción de descargar imágenes grandes que abarquen la zona actual y sus alrededores no es válida: no se permite

generar imágenes de más de 800 pixels
La forma de proceder se basa en permitir movimientos sobre el mapa en sentido horizontal y vertical. Así alrededor de cada porción de mapa visible existen cuatro mapas más que el usuario puede querer ver y que deben de estar precargados (figura 13). Cuando por el movimiento del usuario o simplemente por desplazar el mapa con los dedos, la figura central cambia, se recalculan los índices y se descargan las imágenes necesarias para tenerlas precargadas, todo esllo desde hilos en segundo plano. Estas tareas se activan desde el método *back_interface.changeSlide* que será llamado en cada desplazamiento del mapa.

Para poder trabajar con hilos ha sido necesario modificar la forma de capturar los eventos que GTK+ dispara. Por defecto, la aplicación quedaba bloqueada así que se tuvo que escribir un sustituto de la función *gtk.connect()* a partir de la información encontrada en [16].

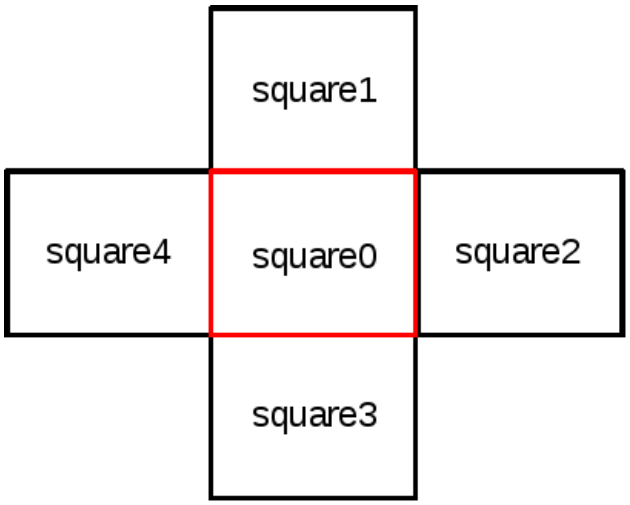

Figura 13: Índices de fragmentos del mapa visibles (en rojo) y no visibles (en negro)

5.6.6. Estructura de la aplicación

La estructura de DónDe cuenta con las siguientes clases principales:

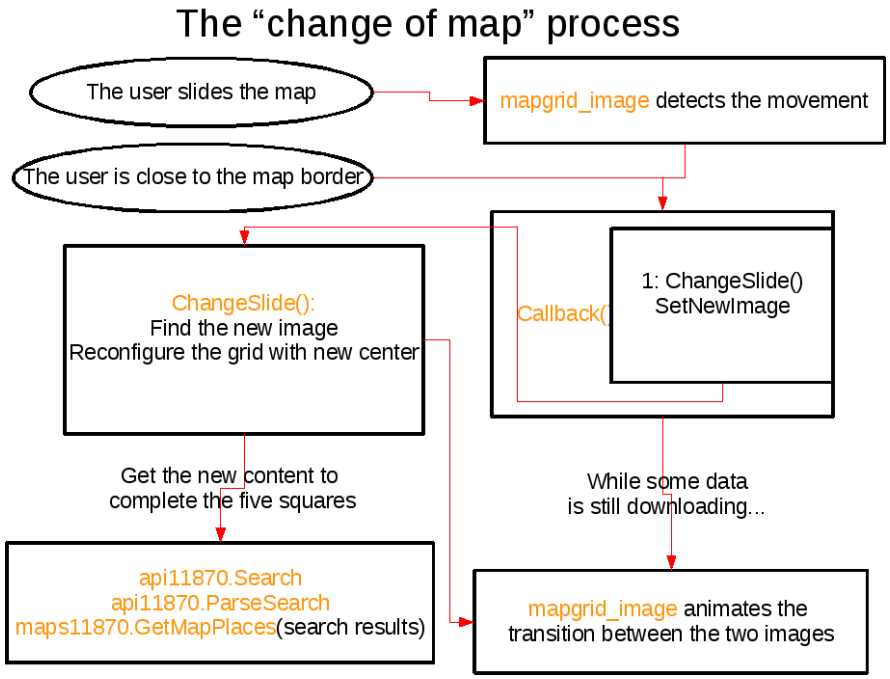

Figura 14: Estructura y funcionamiento de DónDe

mapGridHandler: es la clase principal que se encarga de gestionar la rejilla de mapas que se almacena en memoria. Incorpora los métodos que llaman a las clases de búsqueda en 11870.com y de creación del mapa. Desde aquí se lanzan los hilos que completan la estructura ante cada nuevo refresco del usuario

Square: es cada elemento que forma la rejilla. Cada rejilla se define lógicamente por:

Longitud/latitud: coordenadas del punto central

Imagen: representación gráfica del *Square* en pantalla. Si no tiene valor significa que el “Square” no ha sido iniciado.

Imagen Lógica: contiene la información de los puntos sensibles al tacto dentro de la imagen. Objeto de la clase *logicImageMap*

Resultados de búsqueda: asociados a la cadena de búsqueda actual en el espacio representado por ese cuadrado.

gridStructHandler: es la encargada de manipular la creación y destrucción de *Squares* manteniendo la máxima información posible de los *Squares* actuales para el futuro y añadiendo a los nuevos *Squares* la información básica de longitud/latitud de su punto central.

launchAllThreads/mapGridThreads: cuando el usuario cambia de zona se carga un nuevo mapa, así que su retícula con los mapas de alrededor debe de actualizarse en segundo plano.

Mapas y búsqueda

googleMaps: genera URLs para consultas de distinto tipo y permite agregar *markers* a mapas.

maps11870(googleMaps): hereda la anterior y además permite combinar los mapas de Google con resultados de búsquedas de 11870.

logicImageMap: código para crear y manejar estructuras del tipo lugar /longitud-latitud /pixelx-pixely

base11870: contiene las funciones encargadas de recibir y procesar la información del sitio web 11870.com. Por ejemplo realizar distintos tipos de búsqueda, procesar resultados, procesar elementos individuales, extraer URLS de imágenes, comentarios, etc. Incorpora algunas clases destinadas a la autenticación pero que no han sido completadas debido a que no es necesaria para el proyecto.

scrollImageView, VerticalScrollImage, SquareScrollImage: son las encargadas de realizar el efecto de *slide* en las imágenes capturando el movimiento del dedo y refrescando periódicamente la pantalla. Las últimas heredan de la primera y aportan detección del desplazamiento vertical y desplazamiento horizontal y vertical a la vez.

En la figura 14 se muestra un diagrama de qué eventos y métodos se ven implicados cada vez que se debe de refrescar el contenido de la aplicación.

6. Pruebas

Uno de los aspectos claves de la filosofía de desarrollo de Python se basa en probar cada una de las partes que componen el programa por separado. Para ello existe la condición:

```
if __name__ == '__main__':
```

Que sólo se cumple si el archivo ha sido llamado directamente. De esta forma un mismo archivo puede actuar como librería para otro programa y a la vez como archivo ejecutable, que puede probarse individualmente. Por eso en el código de muchos ficheros del proyecto se encuentra esta sentencia encabezando el último bloque de códio del archivo. Por ejemplo, el fichero *myWidgets.py* contiene el código para mostrar imágenes y suavizar la transición entre ellas que se usa para cambiar de imagen en DónDe. Acudiendo a la parte final de este archivo se encuentra un código de prueba para descargar imágenes de la red y mostrarlas sucesivamente. De esta forma es posible comprobar que el módulo funciona apropiadamente antes de integrarlo en el resto de la aplicación.

Además de este tipo de pruebas se han desarrollado pruebas con cada widget al completo, las denominadas pruebas de interacción que se muestran en las siguientes tablas. Como puede comprobarse, el funcionamiento en la mayoría de los casos comunes es correcto aunque frente a algunos errores se debería mejorar la respuesta que se ofrece al usuario.

6.1. .LRN

Prueba	Resultado
Apertura del widget desde con conexión a Internet	Correcto
Mostrar información desde un servidor modificado para mostrar XML.	Correcto
Mostrar información desde un servidor no modificado	Correcto

6.2. Flickr

Prueba	Resultado
Apertura del widget con conexión a Internet	Correcto
Apertura del widget sin conexión a Internet	Incorrecto: el widget no se inicia y no se proporciona ningún mensaje de error
Apertura del widget por primera vez sin permisos de acceso a Flickr	Correcto: aparece ventana de autenticación
Apertura del widget por primera vez sin archivo de configuración previo	Correcto: aparece menú para introducir opciones
Apertura del menú de Opciones	Correcto
Selección de opción: mostrar favoritas	Correcto: las fotos que se muestran son las favoritas del usuario

Selección de opción: mostrar Recomendadas	Correcto: las fotos que se muestran son las recomendadas por la comunidad
Selección de opción: mostrar según ID de msuario	Correcto: las fotos que se muestran son las del usuario
Selección de opción: cambio del Intervalo de refresco	Correcto

6.3. Twitter

Prueba	Resultado
Apertura del widget con conexión a Internet	Correcto
Apertura del widget sin conexión a Internet	Correcto: aparece mensaje de error.
Apertura del widget por primera vez sin archivo de configuración previo	Correcto: aparece la ventana de configuración
Apertura del menú de opciones	Correcto
Se muestran mensajes juntos con opción activada	Correcto
Cambio de opción: número actualizaciones	Correcto sólo tras reinicio. Si no, se duplican los cuadros de la interfaz
Cambio de opción: juntar Mensajes	Correcto sólo tras reinicio. Si no, se duplican los cuadros de la interfaz
Cambio de opción: nombre de usuario no correcto a correcto	Correcto. Dinámico, no requiere reiniciar.
Envío de actualización	Correcto.

6.4. getPan()

Prueba	Resultado
Inicio vez primera	Correcto
Inicio resto de ocasiones	Correcto
Pulsar botón añadir tarea	Correcto: se abre ventana de add new task
Añadir tarea desde ventana de "add new task" con todos los campos rellenos salvo sonido	Correcto: aparece la tarea de la lista de tareas
Doble click sobre nombre de la tarea	Correcto: aparece descripción y mapa
Click sobre botón de lugares	Correcto: se abre ventana "FavouritePlaces"
FavouritePlaces -> Introducir dirección y pulsar search	Correcto: resultados de búsqueda
FavouritePlaces ->Guardar dirección buscada	Correcto
FavouritePlaces ->Guardar coordenadas	Correcto
Click botón GPS con el estado rojo	Parpadea hasta alcanzar estado verde: muchas veces la sincronización no es posible, o es extremadamente lenta: problema del dispositivo
Click botón GPS con el estado verde	Correcto: pasa a estado rojo
Acercarse a zona donde hay una tarea activada	Correcto: se escucha el sonido y aparece un mensaje de aviso.

6.5. DónDe

Prueba	Resultado
Inicio con conexión a Internet	Correcto
Introducción y búsqueda de dirección	Correcto: mientras se escribe aparecen las alternativas
Introducción del servicio	Correcto
Búsqueda de servicio	Correcto: aparece el nuevo mapa con los servicios correspondientes ubicados
Desplazamiento hacia arriba por el mapa	Correcto: desplazamiento inmediato
Desplazamiento hacia abajo por el mapa	Correcto: desplazamiento inmediato
Desplazamiento izquierda por el mapa	Correcto: desplazamiento inmediato
Desplazamiento Derecha por el mapa	Correcto: desplazamiento inmediato
Click icono de servicio	Correcto: aparece el cartel con el nombre
Click cartel con nombre de servicio	Correcto: aparece la información del servicio
Navegar por fotografías.	Correcto pero cuando las fotografías son de tamaños muy distintos todas se ven con el tamaño de la mas pequeña.
Pulsar flechas para navegar por la lista de comentarios	Correcto
Activar GPS	Correcto: comienza el parpadeo de rojo a verde
Desactivar GPS	correcto

Mostrar posición actual en el mapa	Correcto: aparece un punto en el lugar actual.
Actualizar posición con el movimiento	Correcto: actualización periódica
Cambio de mapa con el movimiento.	Correcto: al acercarse al borde

7. Conclusiones

En este proyecto se han implementado una serie de widgets sobre la plataforma Maemo. El primer objetivo conseguido ha sido establecer una forma de desarrollo rápida utilizando un diseñador gráfico de interfaces disponible para GTK+ (Glade). Así se ha cumplido el objetivo principal: crear una serie de aplicaciones útiles y entregadas en un alto nivel de acabado, cercano al de producción. Sin embargo hay que puntualizar algunos aspectos negativos.
Lo peor viene del lado del GPS. Las aplicaciones que dependen de él se hacen difíciles de usar debido a los largos periodos de sincronización, a los problemas cuando se pierde la señal y hay que volver a sincronizar y sobre todo al consumo de batería que puede terminarse en un par de horas de uso. Estos problemas se producen también en las propias aplicaciones incluidas con la distribución como el software de mapas. También ocasionalmente ocurren otros incidentes, como que el terminal se bloquea cuando se intentan peticiones a Internet sin que haya conexión y pasan periodos de alrededor de un minuto en los cuales no es posible obtener respuesta a ninguna acción. Estos problemas inherentes al dispositivo hacen que de aunque las aplicaciones estén preparadas para ser usadas, difícilmente un usuario final pueda disfrutarlas satisfactoriamente.

A continuación se describen para cada aplicación los objetivos cumplidos y no cumplidos y algunas posibles mejores para el futuro.

7.1. Grado de conclusión de objetivos y trabajo futuro

7.1.1. .LRN

El objetivo marcado era más ambicioso y pretendía tomar información de distintos módulos más complejos: publicaciones en foros, noticias, etc. Sin embargo sin la utilización de la mencionada librería para servicios web de OpenACS, se estableció un método de desarrollo incorrecto con el cual implementar aplicaciones externas era algo complejo y difícil de mantener. Así

pues, la línea de trabajo quedó cerrada con el widget escrito en el que se recupera la información de los usuarios conectados, y el objetivo marcado no puede considerarse cumplido.

7.1.2. Flickr

Se consiguió lo previsto, que era integrar los procesos necesarios para recuperar imágenes almacenadas dentro de Flickr, tanto propiedad del propio usuario de la aplicación como de cualquier otro usuario de Flickr. La funcionalidad buscada era básica y se ha conseguido. Para su distribución tan sólo faltaría afinar el tratamiento de errores y, si se desea, integrar las transiciones fluidas de imágenes desarrolladas para el widget DónDe.

7.1.3. Twitter

Quedan cubiertas las dos funciones básicas de Twitter: recibir dinámicamente las actualizaciones de estado de la gente a la que se sigue y actualizar el propio estado. Sin embargo durante su uso han aparecido algunas posibles mejoras que se podrían incorporar en sucesivas iteraciones. La principal es la de detectar las URLs que se publican en los mensajes de estado y hacerlas clickables para que se abran directamente en el navegador. Esto es útil puesto que Twitter se ha convertido en una herramienta muy usada para compartir enlaces y estar al tanto de las actualizaciones de distintas webs.

La interfaz también puede sufrir un rediseño. Se ocupa demasiado espacio mostrando el nombre del usuario que ha escrito.

También desde el punto de vista técnico sería recomendable cambiar el elemento gráfico usado para mostrar los nombres de quienes han publicado y pasar de una colección de *gtk.labels* a *gtk.TreeView*

7.1.4. getPan()

Es la aplicación más pulida de las que se entregan. Dejando aparte los problemas propios del GPS del dispositivo, cumple con su cometido y con los objetivos marcados en su diseño. En futuras iteraciones se pueden añadir mejoras como una sincronización con servicios de calendario tradicional como Google Calendar de forma que getPan() pudiera descargarse eventos de ahí y ubicarlos en un lugar.

7.1.5. DónDe

DónDe es la aplicación más compleja, funciona correctamente pero adolece de algunos problemas de usabilidad que no han sido corregidos debido a que requieren retocar varias partes. El principal es el tratar de separar los lugares recomendados cuando quedan demasiado juntos. Esto podría hacerse añadiendo algún tipo de zoom o simplemente mostrando todos los lugares disponibles al tocar sobre un lugar con gran concentración de servicios.

A partir de las funciones básicas implementadas se puede avanzar en la aplicación, por ejemplo:

- Añadiendo la opción de mostrar los resultados de las siguientes páginas de búsqueda.
- Permitiendo al usuario evaluar al instante el sitio que está visitando.

A. Manual de uso

A.1. .LRN

Este widget no dispone de ningún tipo de configuración. Sólo son necesarios tres detalles:

- URL del servidor
- Usuario
- Clave

Estos datos han de editarse en un fichero de texto plano con el nombre ".grandientWidget" en el directorio home del usuario. El formato de este fichero es: primera línea *usuario*, segunda línea *pasword* y tercera línea URL.

A.2. Flickr

La configuración y el uso de este widget es muy simple. La ventana de configuración se abre automáticamente la primera vez que se inicia el widget. Posteriormente puede accederse a esta ventana pulsando en "init->subprogram settings" (ver figura 15). Este widget puede mostrar tus fotografías favoritas, las seleccionadas como interesantes por la comunidad Flickr y las fotografías de un determinado usuario (o las tuyas propias). Para ello selecciona la opción apropiada en esta ventana de configuración e introduce el identificador de usuario deseado. Puedes obtener este identificador desde páginas como http://flickrseek.com/?t=gid o http://idgettr.com/

Tras introducir estos datos, hay que completar el proceso de autenticación con Flickr. Toda aplicación que use sus servicios debe de estar autorizada. Esto significa, que este programa puede acceder a tu cuenta de Flickr. Para eso se abre una ventana de navegador que te muestra el mensaje de la figura 16. Para que este widget funcione correctamente has de pulsar en el botón de "Ok, doy mi acuerdo"

A.2.1. Archivo de configuración

La configuración se realiza desde el menú de opciones del programa

```
[general]
refresh_time = 20
tipo = my
url = http://farm3.static.flickr.com/
user = 24397586@N03
configfilename = .flickrPlugin
```

Figure 15: Configuración del wiget para Flickr

Figure 16: Proceso de autenticación en Flickr

A.3. Twitter

Cuando ejecutes este widget por primera vez, verás la ventana de opciones. En esta ventana tienes que introducir los siguientes datos:

- Nombre de usuario.
- Contraseña.
- Número de actualizaciones a mostrar.
- Opción: activar aviso de nuevo mensaje
- Opción: mostrar varios mensajes del mismo usuario juntos. De esta forma junto al nombre de usuario aparecerán los mensajes disponibles. Al tocar varias veces sobre el nombre del usuario se irán sucediendo los distintos mensajes.

Una vez se ha finalizado esta configuración previa, aparecerá la ventana con los mensajes de los *twitteros* a los que sigues. Para volver a mostrar esta ventana de configuración tan sólo basta con acudir al menú inicio->ajustes de subprogramas.

Figura 17: Opciones de Twitter

A.4. getPan()

getPan() es un recordador de tareas basado en espacio. Para añadir una tarea a getPan() es necesario que el lugar en que se va desarrollar la tarea se encuentre previamente en la lista de lugares favoritos, así que agregar una tarea comprende dos pasos:

1. Buscar o introducir manualmente las coordenadas de un lugar o del lugar actual y almacenarlo
2. Agregar una tarea asociándola previamente a un lugar de los almacenados.

Interfaz de la ventana principal

La ventana principal nos muestra las tareas pendientes actuales (figura 18) Tocando dos veces una tarea se puede acceder a su información detallada

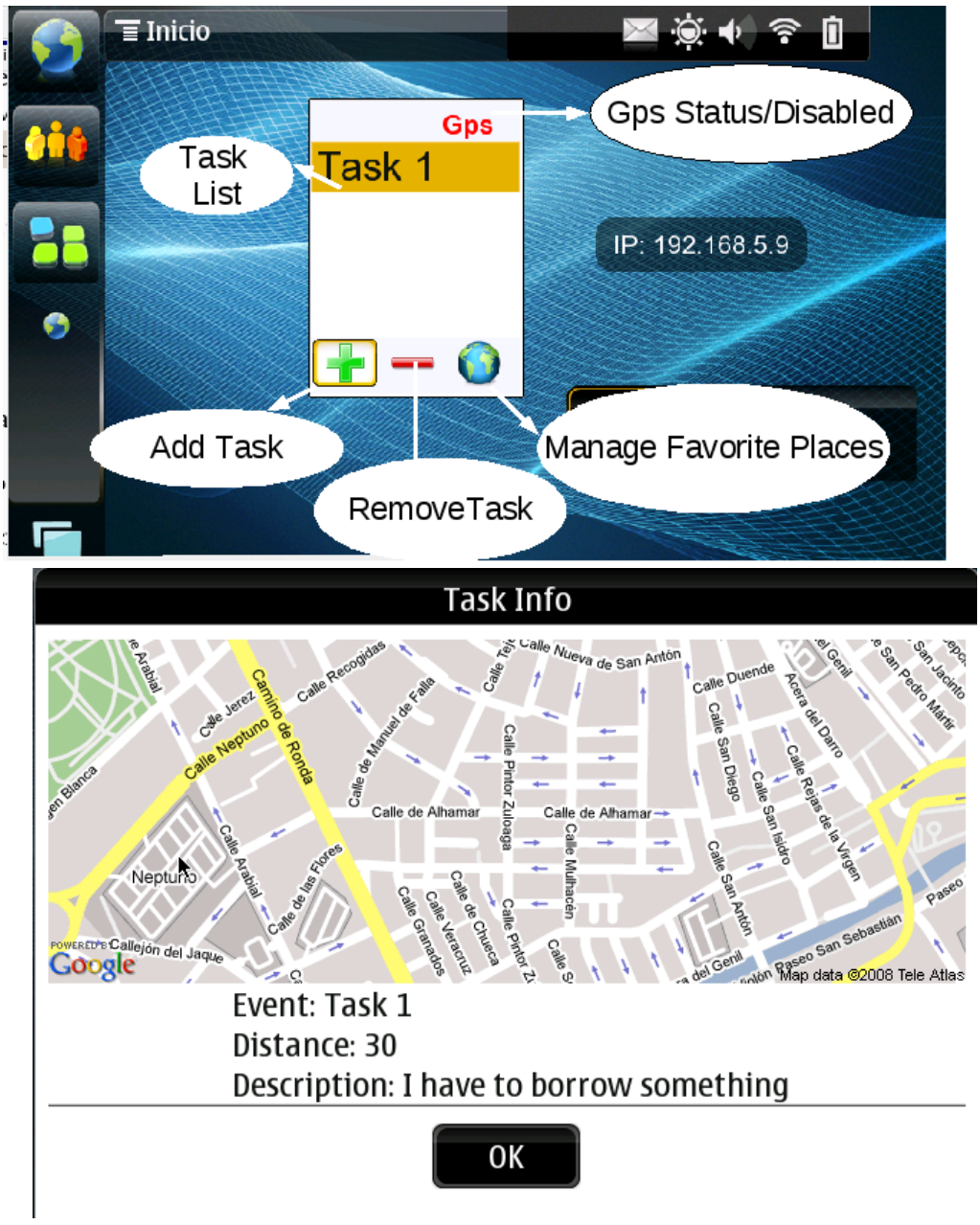

Figure 18: getPan(): ventanas principal y de información de una tarea

Desde aquí es posible añadir nuevas tareas o acudir a la ventana de gestión de lugares. El estado del GPS se muestra en la parte superior. Si la palabra GPS aparece escrita en color rojo significa que el GPS no está activado. Al tocar el botón, comienza la fase de sincronización durante la cual la palabra GPS parpadeará alternando colores rojo y verde. Una vez que el GPS se encuentre sincronizado el color es verde y en la parte superior izquierda aparecerá la distancia hasta la tarea seleccionada.

Añadir tarea

En esta ventana añadimos y configuramos las tareas pendientes. Para acceder a ella hay que pulsar el icono de (+) situado en la ventana principal del widget. A la hora de añadir la tarea es obligatorio incluir la distancia, el

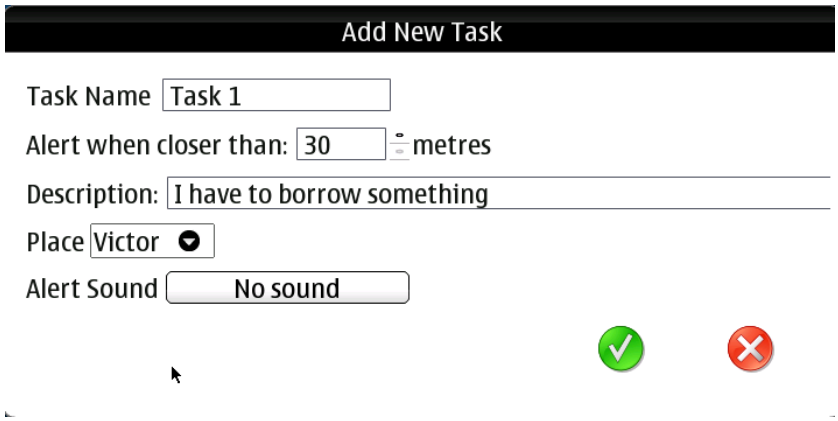

Figura 19: Añadiendo tarea en getPan()

nombre de la tarea y el lugar de la lista de tareas. Como opción también se puede añadir un sonido a la hora de la notificación y una descripción mas detallada de la tarea.

Añadir lugar

Es posible añadir un nuevo lugar a la lista de varias maneras:

- Introduciendo la dirección y pulsando el buscar
- Añadiendo la posición actual: el GPS debe de estar activado y sincronizado.
- Añadiendo manualmente las coordenadas

Una vez seleccionado el método, tan sólo resta indicar el nombre y pulsar "Save". Automáticamente se descargará un mapa de la zona y aparecerá en la lista junto al resto de lugares. Al tocar cualquier elemento de la lista se mostrará el mapa descargado.

Figura 20: Añadiendo lugares a getPan()

A.5. DónDe

DónDe muestra lugares interesantes en un mapa. Cuando el GPS está disponible, es posible centrar el mapa en la posición actual. Cuando el GPS no está disponible la forma de centrar el mapa es escribiendo la dirección.

Una vez que el mapa está centrado, si el GPS está activo el mapa se irá renovando si el usuario se mueve y se sale del mapa actual. También es

posible desplazar el mapa manualmente. Para ello hay que tocar durante un par de segundos cualquier zona libre del mapa y después desplazar hacia arriba abajo, derecha o izquierda.

La parte superior de la aplicación contiene la caja de búsqueda de servicios y el botón de activación del GPS. Si la palabra GPS aparece escrita en color rojo significa que el GPS no está activado. Al tocar el botón, comienza la fase de sincronización durante la cual la palabra GPS parpadeará alternando colores rojo y verde. Una vez que el GPS se encuentre sincronizado el color es verde y en la parte superior izquierda aparecerá la distancia hasta la tarea seleccionada. El tipo de servicio que se busca se introduce en la caja situada arriba a la izquierda y para actualizar el mapa en la posición actual con más servicios basta con pulsar el botón situado junto a la caja.

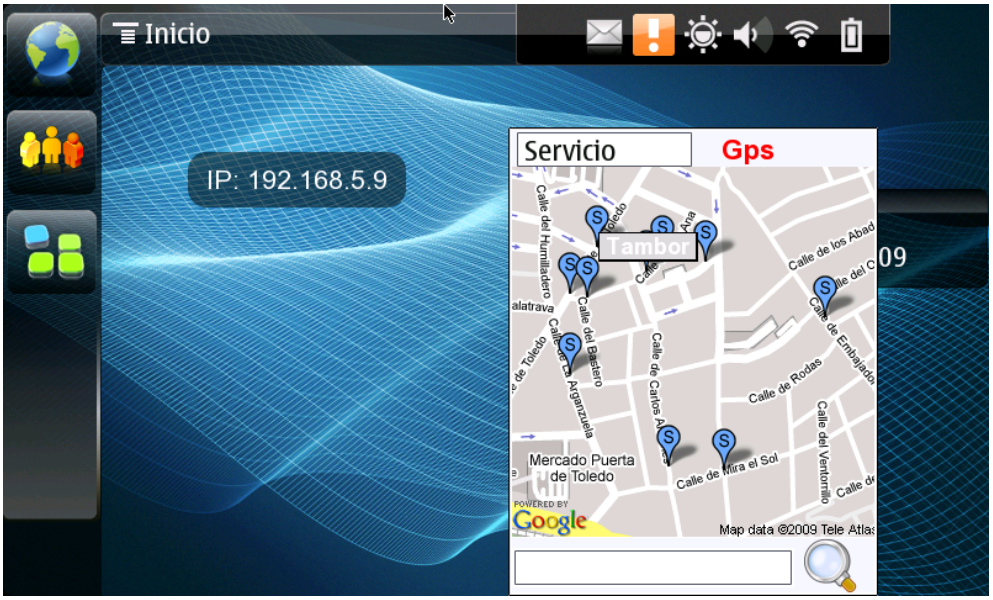

Figura 21: Ventana principal de DónDe

En el mapa aparecen señalados los lugares significativos: al tocar sobre cualquiera de ellos aparece el nombre y al tocar el nombre aparecerá el cuadro de información detallada: descripción y comentarios de los usuarios además de fotografías. Para alternar entre las distintas fotografías basta con pasarlas

deslizando el dedo. Para pasar entre comentarios hay que pulsar en las flechas de avance y retroceso.

B. Manual de instalación.

B.1. Instalación desde archivos fuente

Cada widget contiene al menos dos ficheros:

.py: que contiene hildondesktop.HomeItem class

.desktop: que describe nombre de la aplicación y lenguaje de programación.

Para instalar de forma manual el widget hay que seguir estos pasos:

1. Copiar archivo .desktop a /usr/share/applications/hildon-home/
2. Copiar archivos .py y todos sus sub-directorios a /usr/lib/hildon-desktop/
3. Instalar el soporte para poder cargar widgets escritos en Python:

   apt get install libhildondesktop0 python2.5-hildondesktop

Algunos programas tienen dependencias específicas como:

getPan & DónDe

apt get install geopy

o

sudo easy_install http://geopy.googlecode.com/svn/trunk/

flickrWidget

```
apt-get install python-flickrapi
```

Para disponer del entorno de ejecucción de Python, junto con varias de las librerías comúnmente utilizadas hay que instalar los siguientes paquetes disponibles desde el repositorio *extras*:

```
apt-get install maemo-python-device-env
apt-get install python2.5-libxml2
apt-get install python2.5-hildondesktop
apt-get install python2.5-beautifulsoup
apt-get install python-simplejson
apt-get install python2.5-setuptools
```

Para poder tener acceso root al dispositivo existen varios métodos [29] de los cuales el más sencillo es el de instalar el servidor openssh. Al final del proceso se pregunta por la clave de administrador en el caso de que no se haya establecido previamente.

C. Glosario de términos

Widget: en su primera acepción widget es un elemento que forma parte de una interface gráfica: botones,formularios, barras, etc.
El significado que se le da aquí es el segundo: miniaplicaciones diseñadas para dar fácil acceso a funciones frecuentemente usadas o en general para obtener y mostrar una determinada información de la web.

Plataforma de widgets: programa que permite albergar y ejecutar widgets escritos en lenguajes propios que suelen ser versiones simples de JavaScript y XML.

SmartPhone: teléfono móvil de gama alta que posee potencia suficiente para ejecutar aplicaciones similares a las de un PC: navegador, e-mail, agenda

Scratchbox: parte fundamental del entorno de desarrollo de Maemo. Integra un emulador completo de Maemo dentro de un sistema operativo Linux que hace de host compartiendo parte de sus recursos. Contiene también un compilador cruzado.

GTK+ : (The GIMP Toolkit)
Es una biblioteca gráfica para desarrollar interfaces gráficas de usuario. Núcleo del escritorio GNOME y de Maemo.

Hildon: librerías gráficas que se añaden a las librerías de GTK para proporcionar elementos gráficos adaptados a terminales reducidos.

Hildon desktop home plugin: nombre que reciben, en el escritorio Hildon, los programas que no se ejecutan en una ventana sino que residen "pegados" dentro del escritorio.

Maemo: es el nombre que recibe la plataforma software en la que se basan los dispositivos Internet Tablet de Nokia. Así se habla del Maemo OS (sistema operativo) o del Maemo SDK (entorno de desarrollo)

.LRN: gestor de contenidos orientado a educación.

Locative media*:* información multimedia que está asociada a lugares físicos.

XML: Extensible Markup Language. Es un metalenguaje extensible de etiquetas desarrollado por el World Wide Web Consortium (W3C) de amplia difusión y uso.

CSS: (Hojas de estilo en cascada)
Es un lenguaje usado para definir la presentación de un documento estructurado escrito en HTML o XML.

HTML: lenguaje de Marcas de Hipertexto: usado para describir la estructura y el contenido de un texto de una web.

Web service: conjunto de protocolos y estándares basados en tecnologías web y que sirven para intercambiar datos entre aplicaciones.

Twitter: aplicación web de microblogging que permite publicar mensajes de 140 caracteres y seguir las publicaciones de un grupo de amigos.

Flickr: aplicación web para almacenar y compartir fotografías.

11870: aplicación web para almacenar y evaluar de forma colaborativa servicios como bares, tiendas o restaurantes

Geocoding: proceso de obtener coordenadas geográficas a partir de otro tipo de datos geográficos como direcciones o códigos postales.

GUI: Interfaz Gráfica de Usuario.

Referencias

[1] Nokia N810 webSite http://europe.nokia.com/link?cid=PLAIN_TEXT_508842

[2] Maemo Platform website http://www.maemo.org

[3] Apple MAC OS X widgets http://www.apple.com/downloads/dashboard

[4] Tim O'Reilly, "What Is Web 2.0" http://www.oreillynet.com/pub/a/oreilly/tim/news/2005 /09/30/what-is-web-20.html (accedida Abril 2009)

[5] Richard Wray, "Half world's population will have mobile phone by end of year" The Guardian. http://www.guardian.co.uk/technology/2008 /sep/26/mobilephones.unitednations (accedida Abril 2009)

[6] Reuters "Computers in use pass 1 billion mark: Gartner2." http://www.reuters.com/article/technologyNews/idUSL2324525420080623 (accedida Abril 2009)

[7] Gartner. "Gartner Says Worldwide Smartphone Sales Reached Its Lowest Growth Rate With 3.7 Per Cent Increase in Fourth Quarter of 2008" http://www.gartner.com/it/page.jsp?id=910112 (accedida Abril 2009)

[8] Nokia Widget RuntimeEngine http://sw.nokia.com/id/f8e738e4-cbeb-4b40-b92f-5e8c7cf8dd48/Introducing_Web_Run-Time_v1_1_en.pdf (accedida Mayo 2009)

[9] Alistair Croll. "Nokia, the N810 Tablet & the Long View. http://gigaom.com/2007/11/01/nokia-the-n810-tablet-the-long-view/ (accedida Mayo 2009)

[10] Nokia Linux tablets go 3G http://www.linuxdevices.com/news/NS9093153240.html

[11] http://www.internettablettalk.com/ y www.tabletscene.com (accedida Mayo 2009)

[12] "Shaped widgets? (transparent background)" http://talk.maemo.org/showthread.php?t=16720&page=4 (accedida Mayo 2009)

[13] "Fórmulas de Vincenty" http://es.wikipedia.org/wiki/F%C3%B3rmulas_de_Vincenty (accedida Mayo 2009)

[14] "Programa para conversión entre Pixels y longitud/latitud en grados". http://www.provide.net/~bratliff/adjust.js (accedida Mayo 2009)

[15] "New Python GPS wrapper" http://www.internettablettalk.com/forums/showthread.php?p=168650 (accedida Mayo 2009)

[16] The direct or gtk-lock method http://www.pardon-sleeuwaegen.be/antoon/python/page1.html (accedida Mayo 2009)

[17] Netvibes for iPhone "http://iphone.netvibes.com/"

[18] "ShopSavy" http://www.biggu.com/ (accedida Mayo 2009)

[19] "The Guardian Open Platform" http://www.guardian.co.uk/open-platform (accedida Mayo 2009)

[20] "Parallel Kingdom, Role Playing games" http://parallelkingdom.com/

[21] Lucas Rocha. "Leaving Nokia" http://blogs.gnome.org/lucasr/2008/03/19/leaving-nokia/ (accedida Mayo 2009)

[22] "API de 11870.com" http://groups.google.com/group/api-11870/ (accedida Mayo 2009)

[23] http://wiki.forum.nokia.com/index.php/How_to_create_a_mp3_player_in_maemo
"How to create a mp3 player in Maemo" (accedida Mayo 2009)

[24] http://www.apple.com/itunes/billion-app-countdown/ "Thanks a billion" (Accedida Junio 2009)

[25] http://plusmo.com/featured?t=iphone "Plusmo Widgets for your mobile life" (Accedida Junio 2009)

[26] http://www.mobileburn.com/news.jsp?Id=5986
Ricky Cadden. "Nokia N810 WiMAX Edition discontinued" (Accedida Junio 2009)

[27] http://www.oasis-open.org/committees/wss/documents/
WSS-Username-02-0223-merged.pdf
"Web Services Security". (Accedida Junio 2009)

[28] http://effbot.org/zone/re-sub.htm#strip-html.
Fredrik Lundh. "Converting HTML to Plain Text" (Accedida Junio 2009)

[29] http://wiki.maemo.org/Root_access
"Root access" (Accedida Junio 2009)